2005-06

Electronic attack and sensor fusion techniques for boot-phase defense against multiple ballistic threat missiles

Yildiz, Kursad

Monterey, California. Naval Postgraduate School

NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

ELECTRONIC ATTACK AND SENSOR FUSION TECHNIQUES FOR BOOST-PHASE DEFENSE AGAINST MULTIPLE BALLISTIC THREAT MISSILES

by

Kursad Yildiz

June 2005

Thesis Advisor:	Phillip E. Pace
Co–Advisor:	Murali Tummala

Approved for public release; distribution is unlimited

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704-0188
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.			
1. AGENCY USE ONLY	**2. REPORT DATE** June 2005	**3. REPORT TYPE AND DATES COVERED** Master's Thesis	
4. TITLE AND SUBTITLE: Electronic Attack and Sensor Fusion Techniques for Boost-phase Defense Against Multiple Ballistic Threat Missiles			**5. FUNDING NUMBERS**
6. AUTHOR(S) Kursad Yildiz			
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Center for Joint Services Electronic Warfare Naval Postgraduate School Monterey, CA 93943-5000			**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING /MONITORING AGENCY NAME(S) AND ADDRESS(ES) Missile Defense Agency			**10. SPONSORING/MONITORING AGENCY REPORT NUMBER**
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.			
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited			**12b. DISTRIBUTION CODE** A

13. ABSTRACT (maximum 200 words)

The first objective of this thesis is to investigate the effect of several forms of electronic attack (EA) on the radio frequency (RF) sensors used within a boost-phase ballistic missile intercept system. The EA types examined include noise jamming, chaff, radar cross section (RCS) reduction, and expendable decoys. Effects of the EA methods are evaluated by examining the track position error at the sensor fusion output. Sensor fusion architectures investigated include a weighted average sensor fusion; Kalman-filter-based sensor fusion, and joint probabilistic data fusion architecture. A second objective of this thesis is to extend the single-target, single-interceptor analysis and simulation to a multi-target, multi-interceptor scenario to include the formation of an ellipsoidal gating process to correctly correlate the target measurements with the corresponding track file. We show that the most effective EA is the use of noise jamming followed by a RCS reduction of the missile body. We also show that a properly designed sensor fusion process can effectively mitigate the EA techniques that might be used in a boost-phase intercept scenario.

14. SUBJECT TERMS Boost-phase Ballistic Missile Intercept, Modeling, Simulation, Kalman filtering, Electronic Attack Effects, Proportional Navigation, Radar Cross Section Reduction, RF Sensors, Data Fusion, Decoys, Noise Jamming, chaff, Gating, Ellipsoidal Gate, JPDA, Data Association, Kill Vehicle.			**15. NUMBER OF PAGES** 178
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT** UL

NSN 7540-01-280-5500 Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited

ELECTRONIC ATTACK AND SENSOR FUSION TECHNIQUES FOR BOOST-PHASE DEFENSE AGAINST MULTIPLE BALLISTIC THREAT MISSILES

Kursad Yildiz
Captain, Turkish Air Force
B.S., Turkish Air Force Academy, 1995

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN SYSTEMS ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
June 2005

Author: Kursad Yildiz

Approved by: Phillip E. Pace
Thesis Advisor

Murali Tummala
Co–Advisor

Dan C. Boger
Chairman, Department of Information Sciences

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

The first objective of this thesis is to investigate the effect of several forms of electronic attack (EA) on the radio frequency (RF) sensors used within a boost-phase ballistic missile intercept system. The EA types examined include noise jamming, chaff, radar cross section (RCS) reduction, and expendable decoys. Effects of the EA methods are evaluated by examining the track position error at the sensor fusion output. Sensor fusion architectures investigated include a weighted average sensor fusion, Kalman-filter-based sensor fusion, and joint probabilistic data fusion architecture. A second objective of this thesis is to extend the single-target, single-interceptor analysis and simulation to a multi-target, multi-interceptor scenario to include the formation of an ellipsoidal gating process to correctly correlate the target measurements with the corresponding track file. We show that the most effective EA is the use of noise jamming followed by a RCS reduction of the missile body. We also show that a properly designed sensor fusion process can effectively mitigate the EA techniques that might be used in a boost-phase intercept scenario.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

LIST OF FIGURES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

I would like to thank my wife Filiz and my daughter Ecenur for their patience and support throughout my education at the Naval Postgraduate School.

I would like to thank my thesis advisors, Professor Phillip E. Pace and Professor Murali Tummala, for their help in conducting this research. Also, I would like to thank Professor Bret Michael and the NPS ballistic missile defense team for their valuable ideas.

This work was supported by the Missile Defense Agency.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

A. BOOST-PHASE BALLISTIC MISSILE DEFENSE

> The U.S. effort to develop a non-nuclear anti-ballistic missile defense system gained pace in the early 1990s, and by 1996, the program had become known as National Missile Defense (NDM). NMD consisted not only of the GBI interceptor missile, but also of new ground- and sea-based X-band radars (XBR), a battle management system (BMC^3 - Battle Management Command, Control and Communications), new early warning radars (UEWR - Upgraded Early Warning Radars) and an interface to SBIRS (Space-Based Infrared System) satellites. At that time, it is planned to develop a deployable system until 2000 [1].

The ballistic missile defense system can be divided into three phases: boost-phase, midcourse, and terminal-phase. Figure I-1 shows the three phases of the ballistic missile including, the countermeasures, or electronic attack (EA) deployment, for protection against ballistic missile defense systems. The boost phase lasts anywhere from 180 s to 300 s, depending on the type of propellant used in the missile stages (solid or liquid). At the end of the boost phase, the warhead is propelled into space, where it enters the midcourse phase. Multiple reentry vehicles (RVs) are launched, along with decoys, chaff, and other electronic attack methods to insure a successful flight through space. For the terminal phase, the RVs reenter the atmosphere and proceed to the targeted areas. As is suggested in [2], the only evocative choice for ballistic missile defense is the “layered” approach, which intends to intercept the target missile at every possible trajectory phase.

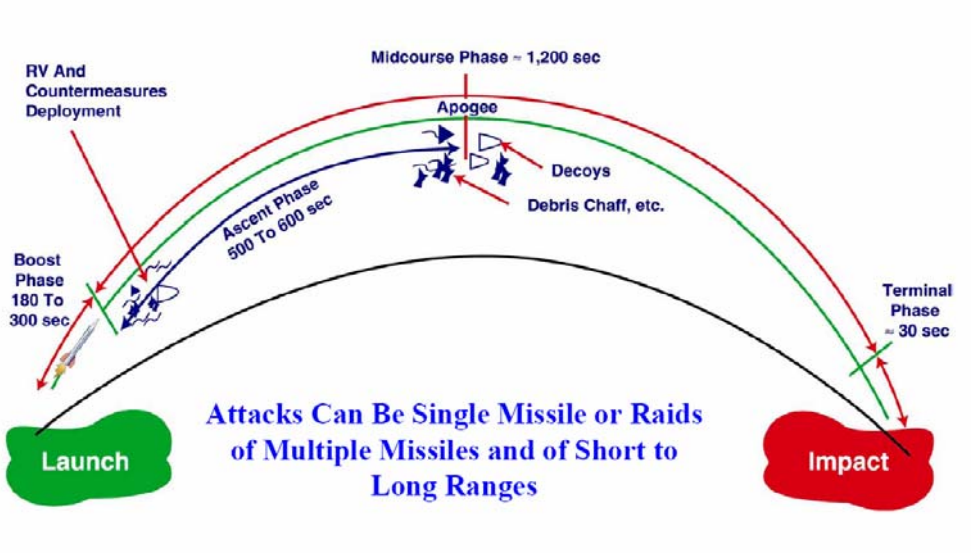


Figure I-1 Three phases of a ballistic missile attack (From [3]).

Defense against the *terminal phase* is difficult, since the defender has to cover all possible aim points. That is, each targeted area must have its own defense system, using multiple kill vehicles for threat interception. As was demonstrated in the Gulf War with the Patriot missile system, unless the interceptor destroys the warhead, the warhead will continue on its path, resulting in the kill of the target [2].

Recently, most of the effort has been focused on intercepting the target in the *midcourse phase*. Midcourse interception is an easier task, since the trajectory and the speed of the target can be estimated accurately. There are, however, drawbacks for engaging the missile in midcourse. These include the possible of intercept debris falling down and landing in friendly territories and the effective use of EA measures (decoys, chaff) to defeat the midcourse interceptors [4].

For defense purposes, the most straightforward way of defeating a ballistic missile is to intercept it in the *boost phase*. Figure I-2 shows a schematic of the possible subsystems involved in a boost-phase defense system. The first step in boost-phase defense is detection of the missile launch by space-based infrared (IR) sensors (since the irradiance is high). After the launch is detected, a number of forward-based radio frequency (RF) sensors can be used to discriminate and track the missile through the boost phase.

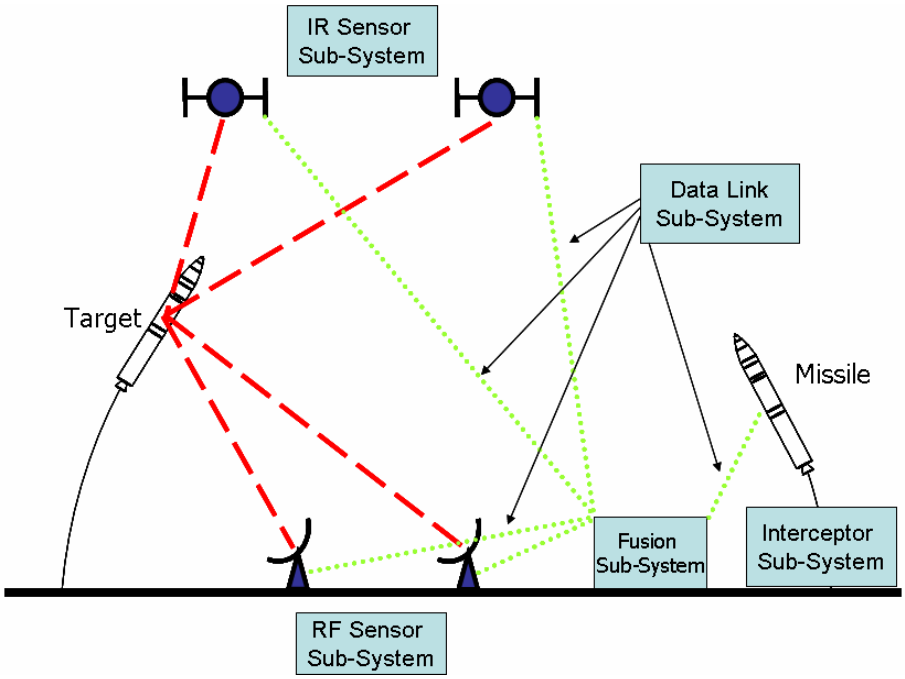


Figure I-2 The subsystem of the boost-phase defense system

As the IR and RF sensors continue tracking the target, the sensor information is transported over a data link to a *fusion sub-system*, which helps refine the estimate of the current and future position of the ballistic target. This refined target position is sent as guidance information to the interceptor, which is assumed to be a sea-based four-stage ballistic missile that carries a *kill vehicle* in its tip. The interceptor flies toward the target, using the guidance information from the sensors, and when in a position that is close enough to the target, launches the kill vehicle in an appropriate direction. After the kill vehicle is launched, it uses its own on-board sensors to navigate in an attempt to intercept and destroy the target.

B. PRINCIPAL CONTRIBUTION

This research investigates the effects of the ballistic missile's EA on the RF sensors within the boost-phase ballistic missile defense system architecture in a multi-target, multi-sensor scenario. Simulations were constructed in MATLAB® to model the various areas of investigation, including correlation, association, and fusion. Motion is simulated in three-dimensional, (3D), ECEF (Earth Centered, Earth Fixed) Cartesian coordinate system.

After the target and the interceptor ballistic missile are delineated, the effects of the EA and the method of mitigating them are explored. The methodology used in this study is a step-by-step approach. In each chapter, the simulation is enhanced to reach the final version.

The single-target simulation from [5] was recoded into functions and details adding the use of electronic attack, state matrix formulation, and drag force. The EA effects on the RF sensors were then investigated quantitatively for a single-target situation. The types of EA used by the missile that were considered include: reductions in the radar cross section (RCS), chaff tactics, deceptive jamming, and expendable decoys. The effects of all EA methods were quantified as: a function of the sensor tracking quality, interceptor guidance, and the miss distance between the interceptor and the target. To mitigate the EA affects, a new Kalman-filter-based sensor-fusion algorithm was investigated for the single-target scenario.

A multi-target, multi-interceptor scenario was developed next. The joint probabilistic data association (JPDA) algorithm (a sub-optimal Bayesian algorithm) and an ellipsoidal gating were included to fuse the sensor tracking results. The reduction in position error is shown for all types of EA considered. Target discrimination is also discussed.

The boost-phase defense simulations were constructed around the following scenario. Two ICBMs are launched from Kilju and Ok'pyong, North Korea, missile bases to hit San Francisco, California. The targets are tracked via sea-based RF sensors. Position data for each target are transmitted to the fusion center with a fixed 10-ms transmission delay. The fusion center processes the information and estimates the targets' positions. The estimated position information is used to control the guidance commands for the interceptors. The interceptors are launched after a 35-s delay (modeling the command-and-control decision time). The interceptors use proportional navigation (PN) guidance to establish collision geometry with the targets.

Each interceptor flies until reaching an optimum position, or burn-out, and then launches its kill vehicle. The kill vehicles (one for each interceptor) and the types of onboard sensors used to navigate to the target for interception are discussed in Chapter IV. Kill vehicle simulation results are shown as a function of the commanded and achieved

acceleration, closure velocity between the kill vehicles and the targets, and the resulting miss distances. Also investigated are the differences between perfect tracking and Gaussian error tracking. The simulation tools developed in this thesis are significant because they allow exploration of other scenarios that may be important; derivation of the optimal solution for each system parameter is also a significant result.

C. THESIS OUTLINE

Chapter II analyzes the effects of the EA on the RF sensor systems. First we address the single-target scenario and the simulation in which this work began. Each EA type is investigated, and the method of deployment in an actual engagement is considered. The results are presented in terms of the tracking (position) error produced by the fusion center, command guidance, and the miss distance.

Chapter III investigates the advanced fusion center algorithm to mitigate the effects of the EA. The Kalman filter and the Bayesian fusion are considered first. Then the system matrices for the Kalman filter are derived. Although the detailed derivations can be found in the related references, the key steps are shown. System noises are explained, and the Kalman algorithm is presented. The results are tested against the considered EA types. Finally, the Bayesian algorithm is analyzed, and the algorithm given in [6] is used.

Chapter IV considers the multi-target scenario and the kill vehicle. Target discrimination, track initializing, correlation, and association are explained in detail. The evolution of the kill vehicle and current state-of-the-art exoatmospheric kill vehicles are discussed. The on-board sensor and the kill requirements are studied. Finally, the simulation results are presented in terms of the tracking ability, produced guidance, closure velocity, and the miss distance.

Chapter V provides concluding remarks.

Appendix A shows the flowchart of the final simulation. Appendix B is a complete listing of the code. Appendix C gives the read-me files for maintenance of the simulation.

THIS PAGE INTENTIONALLY LEFT BLANK

II. ELECTRONIC ATTACK EFFECTS

This chapter presents the effect of the EA on the RF sensors used within the Intercontinental Ballistic Missile (ICBM) boost-phase intercept system. The interception of an ICBM can be considered as a "sequence of successful tasks: detecting and classifying the threat missile, predicting the threat trajectory, discriminating the target from clutter and unlikely echoes, tracking the target, acquiring the target for interception, intercept, and kill assessment" [2]. If one of these tasks is not successful, the intercept is in jeopardy. The EAs used by the ICBM are intended to cause an intercept attempt to be unsuccessful. The EA investigated in this thesis are listed in Table II-1.

Table II-1 EAs That Can Degrade Defense Effectiveness

Radar absorbing materials for reducing the radar cross section (RCS) of the target
Radar decoys to lure the radar tracking
Coated glass fiber chaff for obscuring the target echo
Deception jamming for forcing radar to lose the track

A. PREVIOUS SCENARIO AND SIMULATION

In this section, the scenario and the simulation described in [5] and [6] are reviewed. As described, an ICBM is launched from North Korea from Kilju Missile Base to hit San Francisco, California. The target is tracked via two ground-(actually sea)-based RF sensors and two space-based IR sensors. The target position data are transmitted to the fusion center in an ECEF Cartesian coordinate system. The fusion center combines the position information and refines the target's estimated position. Once a firing solution is obtained, the interceptor is launched toward the target ICBM. The estimated position information is sent to the interceptor via a communication link. The interceptor uses the information to derive the guidance force required for the intercept. The interceptor and the target ICBM establish the collision geometry conditional on the PN guidance rules. The interceptor flies until crossover with no kill vehicle being used.

To evaluate the scenario, a 3D ballistic-missile interception simulation was developed in [5]. The dynamics of the ballistic missile and a 3D motion algorithm depend on total mass, propellant mass, and the specific impulse in the gravity field of a perfect-sphere, non-rotating-earth is modeled [5].

For the target ICBM, both [5] and [6] used a three-stage missile with the total mass and dimension of each stage being the same as the U.S. Peacekeeper missile, with 85 percent of the total mass of each stage being the propellant mass. Each stage was assumed to be using a fuel with a specific impulse of 300 s and a burnout time of 60 s. The total boost phase took three minutes. The target was assumed to be carrying a payload of 5,000 lbs.

For the interceptor ICBM, three generic models ware investigated, of which the third was the best. The interceptor was a three-stage missile, and the total mass and dimension of each stage also bears a resemblance to the U.S. Peacekeeper missile, but with 95 percent of the total mass of each stage being the propellant mass. Each stage was assumed to be using a fuel with a specific impulse of 300 s and a burnout time of 60 s. The total boost phase takes three minutes. This missile was assumed to be carrying a payload (kill vehicle) of 1,500 lbs (although the kill vehicle was never used).

Both [5] and [6] also explored the specifications for the space-based IR sensors used to provide the initial launch detection. Detailed models of the missile plume signature were also examined. The IR sensors were designed to have a uniform error performance within their instantaneous field of view for tracking. The RCS of the target ICBM and definition of the RF sensor parameters (given in Table II-3 in this work) were derived in [5]; the error performance of the sensors was quantified depending on the target RCS value.

B. TO BE SEEN OR NOT TO BE SEEN

ICBM attack is considered to be detected via electronic systems, such as a space-based electro-optics/IR systems or a ground- or air–based RF systems. Both of these systems work by using the target's signatures, such as its minimum resolvable temperature difference or its RCS. The ICBM must *reduce* these signatures in order not to be seen or detected.

In this section, we deal with the RCS reduction to reduce the chance of being detected by the radar.

1. Radar Cross Section

Radar performance is directly related to the RCS of a target in the radar range equation [7]

$$R_{\max} = \left[\frac{P_t G_t^2 \lambda^2 \sigma n E_i(n)}{(4\pi)^3 k T_0 B_n F_n (S/N)_1} \right]^{1/4} \quad \text{m} \tag{2.1.1}$$

where P_t is radiated power, G_t is gain of the radar antenna, λ is wavelength, σ is RCS of the target, n is number of pulses in a train, $E_i(n)$ is integration efficiency, $k = 1.38 \times 10^{-23}$ J/deg is Boltzmann's constant, $T_0 = 290^o$ K is standard temperature, B_n is the bandwidth of the receiver, F_n is the noise factor, and $(S/N)_1$ is the signal-to-noise ratio for a single pulse.

The only parameter independent of the radar system is the RCS of the target. As shown in (2.1.1), the range can be halved when the RCS is decreased by 1/16$^{\text{th}}$ of its original value. The RCS can be defined as [8]:

$$\sigma = \frac{\text{Power Reflected to receiver per unit solid angle}}{\text{Incident power density}/4\pi} \quad \text{m}^2$$

In terms of the incident and scattered electric field intensities, $\vec{E}_i$ and $\vec{E}_s$,

$$\sigma = \lim_{R \to \infty} 4\pi R^2 \frac{\left|\vec{E}_s\right|^2}{\left|\vec{E}_i\right|^2} \quad \text{m}^2 \tag{2.1.2}$$

where R is the range to the target. The RCS is affected by the frequency and polarization of the incident field and the target aspect relative to the radar. Figure II-1 shows the typical average values of the RCS.

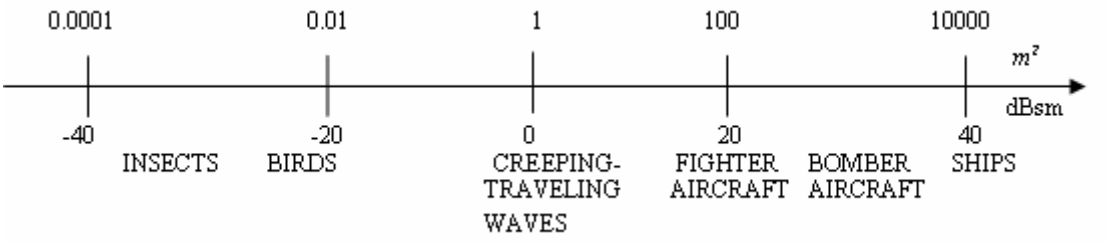


Figure II-1 Typical values of the RCS (After [8])

The RCS of the three-stage generic ICBM target was predicted by [5] by using *POFACETS* [9]. The model of the generic missile is created by using triangular facets. For the target model, the parameter is taken from the US Peacekeeper missile because of its long-range capability and the availability of detailed open-source literature about the missile. The best frequency band is X-Band (10 GHz); that is optimum for the calculation [10]. The RCS model was built using the original dimensions of the missile. It was assumed to be constructed of aluminum and titanium alloy, which is heat resistant. The conductivity was assumed to be 2×10^7 S/m. The standard deviation of the surface roughness was set to 0.3 mm. The model used in the program is shown in Figure II-2.

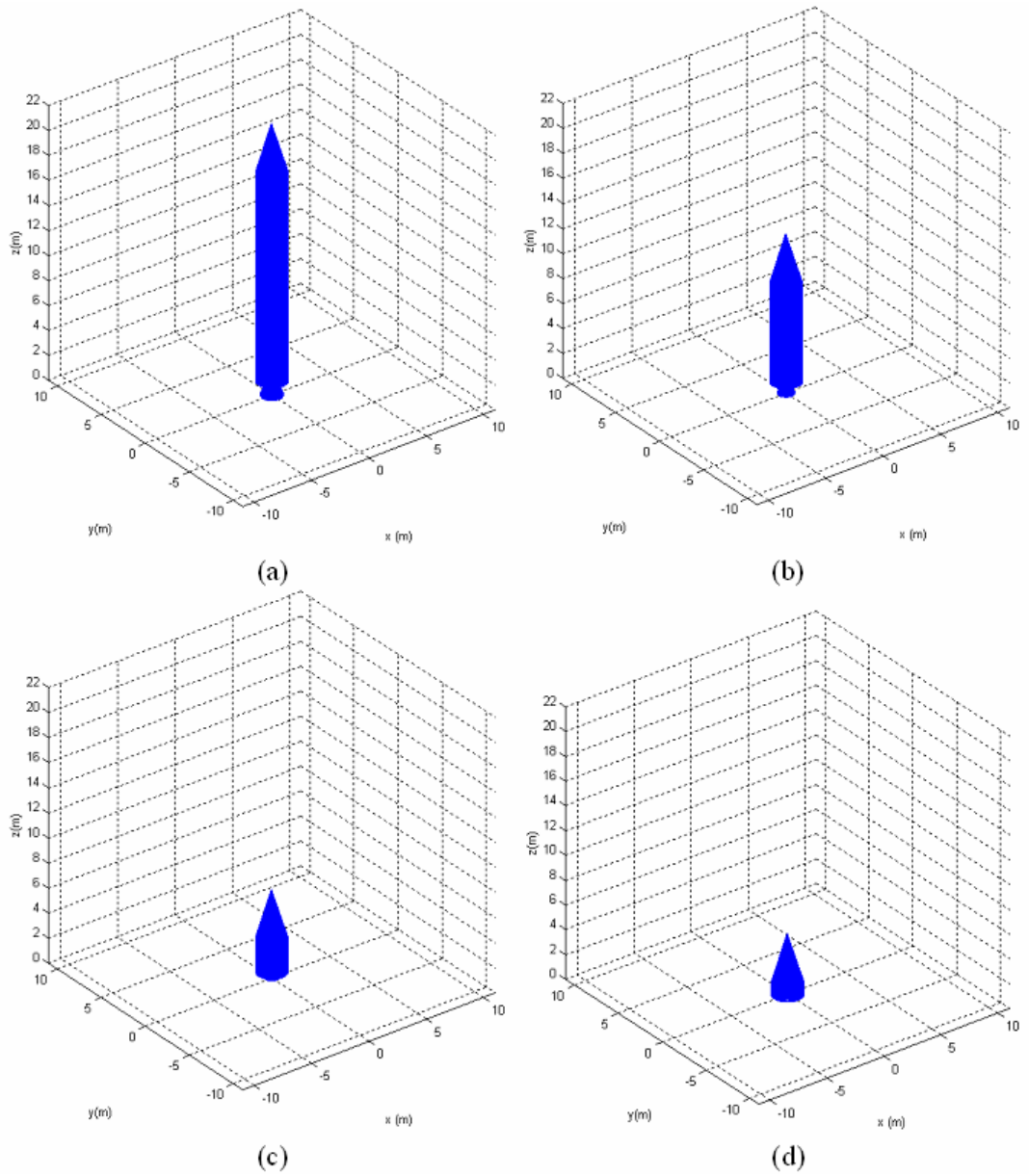


Figure II-2 Full-Scale Models of the missile at different Stages: (a) Stage 1, (b) Stage 2, (c) Stage 3, (d) Payload. (From [5])

2. RCS Reduction

When talking about the reduction of the RCS, four approaches can be considered [8]: 1) target shaping, 2) passive cancellation, 3) active cancellation, and 4) materials selection and coating. Each type has its advantages and trade-offs.

a. Shaping

Generally, the shaping is considered to be the first and most important step to RCS reduction. Shaping changes the geometry of the platform so that the electromagnetic waves do not reflect back to the source. The best example for this method is a plate having dimensions of $a\lambda$ m $\times a\lambda$ m. Even if the normal physical area is the same, the diamond-shaped plate has a smaller specular RCS from the front aspect than a normal square plate. The shape controls the principal plane reflection, so the radar sees a reduced RCS from the front side. The Lockheed *F-117A* and Northrop *B-2* have this kind of facet, which reduces their RCS from the principal aspect. Neither of these aircraft is supersonic. The trade-off for the use of shaping is reduced aerodynamic capabilities and increased cost.

Reduction of the RCS cannot be done for the entire aspect angle of the platform because there will always be a viewing angle from which the radar can see the platform in normal incidence. Because we are considering an ICBM, the aerodynamic performance is very important. But that aspect of the ICBM is beyond the scope of this thesis, so we do not investigate it.

b. Passive Cancellation

This method is better known as impedance loading [8]. It can be done by introducing a second reflector to cancel out the original one, by matching the amplitudes and phases. This process can be done for relatively simple platforms whose reflection properties are well known. However, the radar parameters of the target and its orientation also must be known well enough to apply this method. Although this method can be used to supplement any shaping as well, we cannot be sure about the aspect of our missile and thus this method also is not examined.

c. Active Cancellation

This method involves modifying and processing the received radar signal and then retransmitting it to cancel out the reflection [8]. To do this, we need to know everything (angle of arrival, phase, pulsewidth, PRF, etc.) about the source radar. This is similar to active jamming techniques that use digital radio frequency memories (DRFMs) for storing and repeating the radar signal. This is a very challenging process; and because the demands in applying this method are always considerable, it is not practical for large

targets such as ICBMs. In some cases, knowledge of the target radar is not enough; the platform's own RCS value should be known as well for all aspects.

d. Material Selection and Coating (Radar Absorption Material)

Using radar absorption material (RAM) is another way to reduce the RCS. The main purpose of this technique is to use material to cover the platform, in order to absorb or attenuate the incidence waves so that no, or few electromagnetic reflected waves are sent back to the radar. For this purpose, composite materials are commonly used.

The amount of loss in the material is determined by the loss tangent (δ_ε and δ_μ) of the material. The loss tangent is dependent on the imaginary part of the permittivity and the permeability [11]. There are two theorems about the absorber.

Theorem 1 [8]: "if a plane electromagnetic wave is incident on a body composed of material such that $\mu/\mu_0 = \varepsilon/\varepsilon_o$ (which states that $\mu_r = \varepsilon_r$) at each point, then the back scattered field is zero, provided that the incidence direction is parallel to an axis of the body about which a rotation of 90 deg leaves the shape of the body, together with its material medium invariant."

Theorem 2 states that "if a plane wave is incident on a body composed of a material such that the total field components satisfy the impedance boundary condition and if the surface is invariant under a 90 deg rotation, then the back scattered field is zero if the direction of the incidence is along the axis of symmetry and $Z_s = 1$" [8].

Both theorems are derived from Maxwell's equations. Now, we need to select the material that creates the smallest reflected waves.

Composite absorbers are produced by using existing materials and varying their permittivity and permeability [8]. Both of which are generally complex numbers. Common dielectric materials used for absorbers, such as foams, plastics, and elastomers, have no magnetic properties. Magnetic materials, such as ferrite, iron, and cobalt-nickel alloys, are used to alter the permeability of the materials. High dielectric materials, such as carbon, graphite, and metal flakes, are used to modify the dielectric properties [11].

To apply the RAM, there are two primary approaches: the "*Salisbury screen*" and the "*Dallenbach layer*". In this study, the Dallenbach layer is investigated, which allows us to construct a broadband canceller by applying multiple layers if necessary. Before considering the material, several questions must be addressed to determine the material requirements. Questions to be addressed are [11]:

- Which frequency bands need to be covered?
- Should coverage be applied for the entire band or for a specific frequency?
- Is the RAM used to attenuate all reflections or just for traveling waves?
- Will the RAM be used in a closed or open environment?
- What kind of environmental force will affect the RAM: salt, water, ozone, oxygen, ultraviolet light, fuels, oils, chemicals, nuclear, stack gases, heat, etc.?
- What mechanical stresses will be placed on the RAM: vibration, thermal shock, elongation, wind, etc.?
- What is the expected lifetime of the RAM?

When all these questions are answered, we will know the electrical and physical performance characteristics of material, which include its temperature, environmental, and mechanical properties. In our case, the physical and the electrical performance are equally important.

After answering the questions, we conclude that the material should work at X-band, especially 10 GHz. Note that intelligence information about the target radar is important here. We assume that the intelligence information about the interceptor radar is sufficient. Because the ICBM flies in the dense atmosphere for at least 100 km during the boost-phase, which approximately takes 120 s according to the previous simulation, we need a material that can resist the temperature caused by the drag of the air. The layer must be as thin as possible to reduce extra weight that would cause degradation of the ICBM's performance. The life expectancy of the layer is at most 4-5 minutes, depending on the missile boost-phase.

Finally we may need a composite material containing titanium, ferrite, carbonyl iron, silicon rubber, and nickel to satisfy all these needs. Our backing material is an alloy of titanium and aluminum, which has an assumed conductivity of 2×10^7 S/m (rela-

tive permeability and permittivity are 1). For 10 GHz, we can find the conductivity of the alloy [12]. The intrinsic impedance of the medium is defined as

$$\overline{\eta} = \sqrt{\frac{j\omega\mu}{\sigma + j\omega\varepsilon}} \tag{2.1.3}$$

where $\omega = 2\pi f$, and σ is conductivity, and μ and ε are the permeability and the permittivity of the medium, respectively. The intrinsic impedance of the medium is complex so long as the conductivity is not zero. The phase angle of the intrinsic impedance indicates that the electric field and the magnetic field are out of phase. By applying wave equations to the material, we conclude that there are three types of material on the earth:

perfect dielectric: $\sigma/\omega\varepsilon << 1$,

imperfect dielectric: $10 > \frac{\sigma}{\omega\varepsilon} > 0.1$, and

good Conductor: $\sigma/\omega\varepsilon >> 1$

where $\frac{\sigma}{\omega\varepsilon} = \frac{\varepsilon''}{\varepsilon'} = \delta_{\varepsilon}$ is the loss tangent of the medium, and ε'' and ε' are imaginary and real parts of the permittivity, respectively. By using the above properties, we find that

$$\frac{\sigma}{\omega\varepsilon} = \frac{2\times10^{7}}{2\times\pi\times10\times10\times^{9}\,8.854\times10^{-12}} = 3.6\times10^{7} >> 1$$

Our alloy seems to be a very good conductor and can be accepted as a perfect electric conductor (PEC) for our study. Now we need to design a composite material over the PEC material for the reduction.

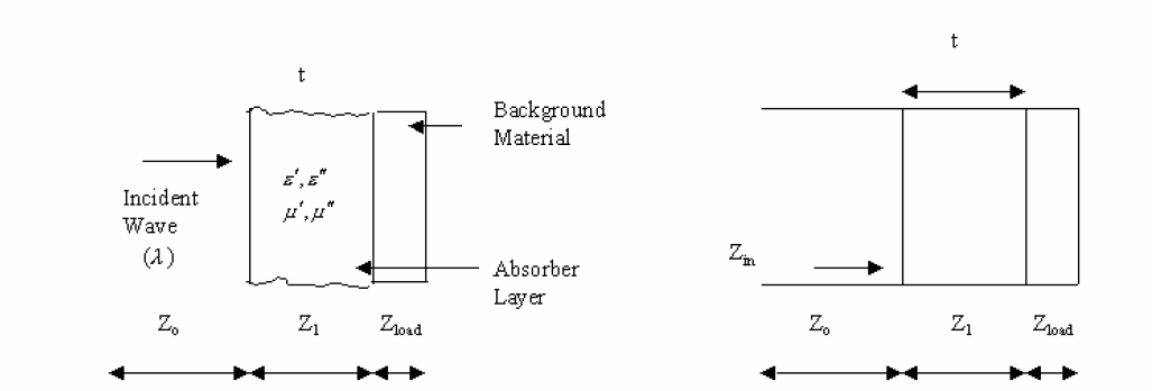


Figure II-3 Specular Reflection (From [13]).

The reflection coefficient is the ratio between incident and the reflected electric field and is defined as

$$\Gamma = \frac{Z_{in} - Z_0}{Z_{in} + Z_0} \tag{2.1.4}$$

where Z_0 is the *characteristic impedance*, and Z_{in} can be defined using transmission line equation shown in Figure II-3

$$Z_{in} = Z_1 \frac{Z_{load} + jZ_1 \tan(\beta t)}{Z_1 + jZ_{load} \tan(\beta t)} \tag{2.1.5}$$

where Z_1 and Z_{load} come from the absorption material and the backing material, respectively. The propagation constant β, Z_1, and Z_{load} are defined in [12] as

$$\begin{aligned} \beta &= \frac{2\pi}{\lambda}\sqrt{(\varepsilon_r' - j\varepsilon_r'')(\mu_r' - j\mu_r'')} \\ Z_{load} &= \sqrt{\frac{\mu_o(\mu_r' - j\mu_r'')}{\varepsilon_0(\varepsilon_r' - j\varepsilon_r'')}} \\ Z_1 &= \sqrt{\frac{\mu}{\varepsilon}} \end{aligned} \tag{2.1.6}$$

If we normalize (2.1.5) with Z_0, we have a transcendental equation with six unknowns, which can only be evaluated numerically. This can be done by assigning numerical values to four of the six values and using a computer-implemented complex-

root-finder algorithm to find the numerical values of the remaining two values that satisfy the condition [14]. MATLAB and MATHCAD can be used to solve this equation [13]. The resulting equation is

$$Z_s = \frac{Z_1}{Z_0}\frac{\frac{Z_{load}}{Z_0} + j\frac{Z_1}{Z_0}\tan(\beta t)}{\frac{Z_1}{Z_0} + j\frac{Z_{load}}{Z_0}\tan(\beta t)}$$

$$Z_s = \sqrt{\frac{(\mu_r' - j\mu_r'')}{(\varepsilon_r' - j\varepsilon_r'')}}\frac{\sqrt{\frac{1}{(\varepsilon_r' - j\varepsilon_r'')}} + j\sqrt{\frac{(\mu_r' - j\mu_r'')}{(\varepsilon_r' - j\varepsilon_r'')}}\tan(\frac{2\pi}{\lambda}\sqrt{(\varepsilon_r' - j\varepsilon_r'')(\mu_r' - j\mu_r'')}t)}{\sqrt{\frac{(\mu_r' - j\mu_r'')}{(\varepsilon_r' - j\varepsilon_r'')}} + j\sqrt{\frac{1}{(\varepsilon_r' - j\varepsilon_r'')}}\tan(\frac{2\pi}{\lambda}\sqrt{(\varepsilon_r' - j\varepsilon_r'')(\mu_r' - j\mu_r'')}t)} \quad (2.1.7)$$

For our case, the backing material is PEC so $Z_{load} = 0$. In this case, the resulting equation is

$$1 = j\sqrt{\frac{(\mu_r' - j\mu_r'')}{(\varepsilon_r' - j\varepsilon_r'')}}\tan(\frac{2\pi}{\lambda}t\sqrt{(\varepsilon_r' - j\varepsilon_r'')(\mu_r' - j\mu_r'')}) \quad (2.1.8)$$

The six unknown can be reduced to four by normalizing with t/λ as follows:

$$a = \frac{t}{\lambda}\varepsilon_r',\ b = \frac{t}{\lambda}\varepsilon_r'',\ x = \frac{t}{\lambda}\mu_r',\ y = \frac{t}{\lambda}\mu_r'' \quad (2.1.9)$$

In terms of new parameters, our transcendental equation is now

$$1 = j\sqrt{\frac{x - jy}{a - jb}}\tan(2\pi\sqrt{(a - jb)(x - jy)}) \quad (2.1.10)$$

By solving this equation, we can create the Universal design chart in Figure II-4. By using that chart, we can predict the material electrical properties. For a frequency of 10 GHz, $\lambda = 0.03$ m. The layer thickness must be as low as 1 mm so that $t/\lambda = 1/30$ to get a good cancellation for a relatively small amount of extra weight.

We assume that the composite material consists of titanium, ferrite, carbonyl iron, silicone rubber, and nickel. Most metals have $\varepsilon_r \simeq 1$ and for most nonmagnetic materials $\mu_r \simeq 1$. Titanium and nickel have $\varepsilon_r = 1$. Carbonyl iron has

$\varepsilon_r = 4 - j0.23$, with 50% of its mass being C: Fe [15]. Silicone rubber has $\varepsilon_r \simeq 3 - j0.225$ [16], and finally ferrite-50 has $\varepsilon_r \simeq 21 - j13.86$ [17]. By using all these materials in sufficient amounts, the composite material should be able to reach the final permittivity. Let us assume that the material property changes linearly depending upon the ingredients. Using the Microsoft Excel "solver add-in" to find the percentile of the material needed, we optain Table II-2 that displays the new material properties. Note that the objective is to maximize the loss tangent, subject to reaching a small density amount in order to minimize extra weight.

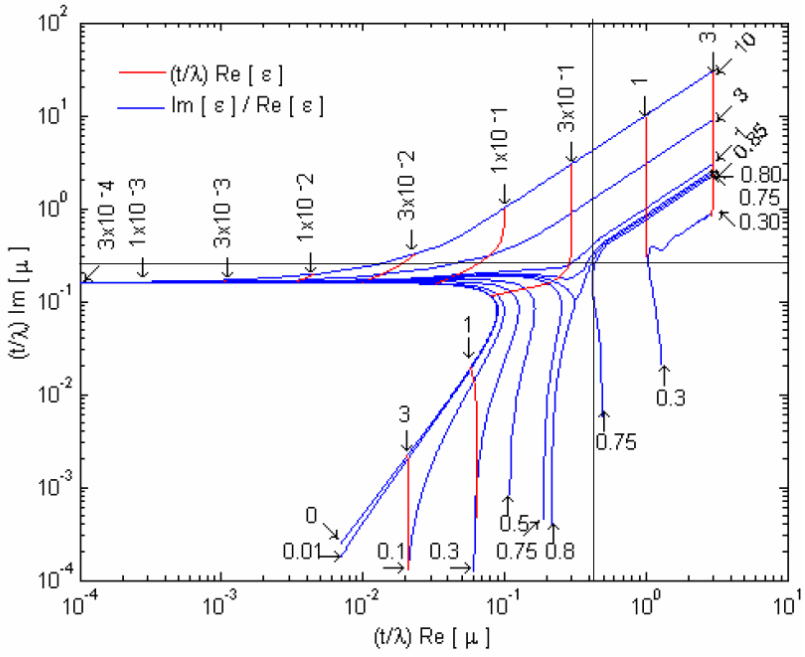


Figure II-4 Universal Design Chart for zero specular reflection absorber layer (PEC) (After [13])

Table II-2 New Material properties

Relative Permittivity	*Ferrite-50*	*Carbonyl Iron*	*Silicone Rubber*	*Titanium*	*Nickel*		*Total Density*
	0.49	***7.87***	***0.98***	***4.5***	***8.9***	**Density (gr/cm^3)**	1.83
Real	21	4	3	1	1		
Imaginary	13.86	0.23	0.225	0	0	**TOTAL**	
						Found	**Needed**
Percentile	0.6	0	0.2	0.1	0.1	1	1
New Material		AIM	**Loss Tangent**				
Real	13.4	12	0.623955				
Imaginary	8.361	12					

Our computation concludes that the composite material needs to have 60 percent ferrite-50, 20 percent silicone rubber, 10 percent titanium and 10 percent nickel. With this composition, the complex relative permittivity will be

$$\varepsilon_r = 13.4 - j8.361 \tag{2.1.11}$$

As shown in the table, the new composite material has a density of $1.83\,\mathrm{gr}/\mathrm{cm}^3$. The extra weight that the layer adds to the missile is found by determining the volume. The surface area of the missile can be calculated to be 1.4×10^6 cm^2 (surface area of a cylinder is $2\pi rh$ and that of a cone is πrl). For a thickness of 1 mm, the final volume will be $1.4\times10^5\,\mathrm{cm}^3$. This results in an extra weight of 256.2 kg, far less than the total weight of the missile (86,636 kg) and insignificant for our simulation.

Now, we must determine the permeability of the layer. By applying (2.1.11), the values "a" and "b" and the loss tangent are found to be

$$a = \frac{t}{\lambda}\varepsilon_r' = \frac{1}{30}13.4 = 0.447$$
$$b = \frac{t}{\lambda}\varepsilon_r'' = \frac{1}{30}8.361 = 0.2787$$
$$\delta_\varepsilon = \frac{\varepsilon_r''}{\varepsilon_r'} = \frac{8.361}{13.4} = 0.623955$$

By using Figure II- 4 (depicted as a line) or using the MATLAB "solve" built-in function, we calculate the μ_r' and μ_r'' values as

$$x = \frac{t}{\lambda}\mu_r' = 0.44 \Rightarrow \mu_r' = 13.17$$
$$y = \frac{t}{\lambda}\mu_r'' = 0.277 \Rightarrow \mu_r'' = 8.31$$

Finally, we determine the permeability of the composite material as

$$\mu_r = 13.17 - j8.31 \tag{2.1.12}$$

By using (2.1.11) and (2.1.12) and editing "utilities" in the *POFACETS* [9], the new material can be created to apply over the ICBM. After applying the material, Figure II-5 is a plot of the missile with the new RCS values that is also compared to the original values.

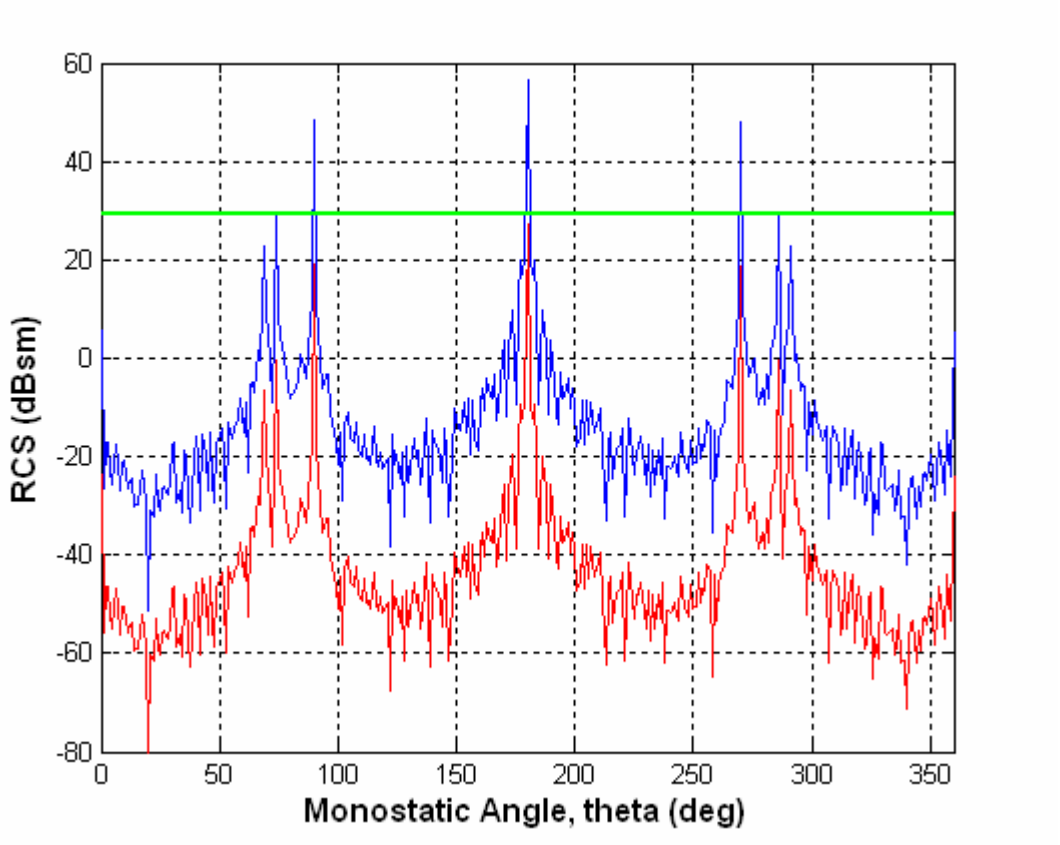


Figure II-5 Comparison of RCS for stage-1. The red line represents the reduced RCS. The green line represents the average reduction.

As shown in Figure II-5, the composite material achieves an average reduction of 29 dBsm in the RCS of the missile stage. Figure II-5 shows only stage-1 results and demonstrates the significant change in the RCS. The other stages have the same pattern of reduction. The realization of such composite material, a method to achieve a 1 mm thickness on the missile, and a cost analysis are all beyond the scope of this thesis. Nonetheless, they comprise a practical goal that may be achieved in the next five years. All such missile applications require advanced systems engineering.

3. Effect of RCS Reduction

Figure II-6 plots the magnitude of the position error when a RCS reduction is used in the simulation. Note that, when using normal RCS values, the average position error is about 70 m, with a maximum error of 140 m [5]. As shown in Figure II-6, the average position error introduced by a reduced RCS is 1500 m; the maximum position error is

4500 m. The original simulation used a guidance update time of 0.15 s. When that update time is retained, the interceptor is responsive, cannot fly more than 5 miles and crashes. In Figure II-6, the update time used to collect the data is increased to 2 s.

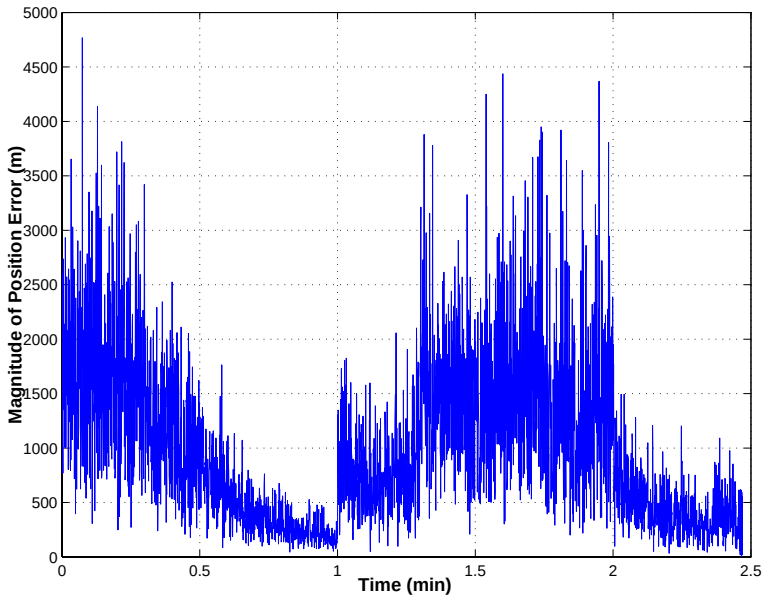


Figure II-6 Position error introduced by fusion system when the RCS is reduced.

When a large difference occurs in the average sensed position of the target at each sample time, the navigation unit sends large acceleration commands. Figure II-7 shows the closure velocity for the interceptor (the data update interval is 2 s).

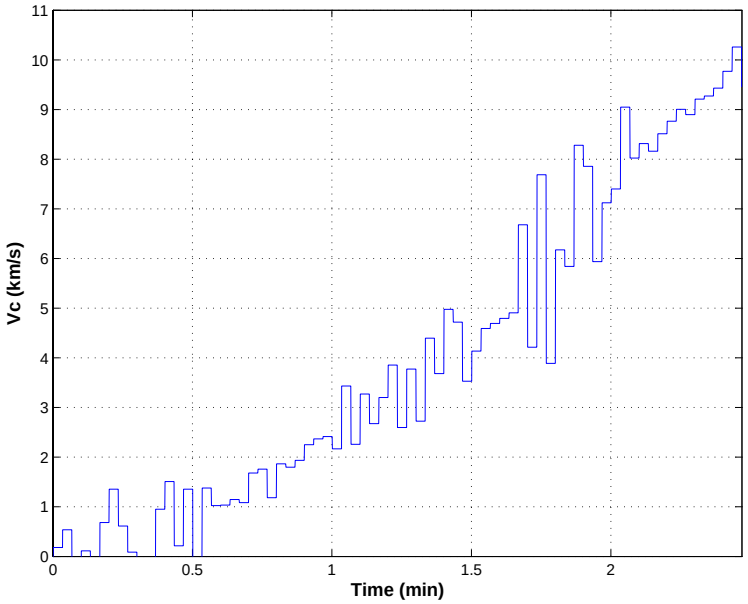


Figure II-7 Closing velocity versus time during the intercept of a reduced-RCS target. (The data update time is increased to 2 s.)

The fusion algorithm used in Figure II-7 is an arithmetic averaging method that was used in the original version of the simulation. Better performance is obtained by using a Bayesian and Kalman filtering fusion technique, which will be investigated in Chapter III. Figure II-8 shows both the commanded and the achieved lateral acceration obtained by the missile throughout the flight (the update time is 2 s, using the original fusion algorithm).

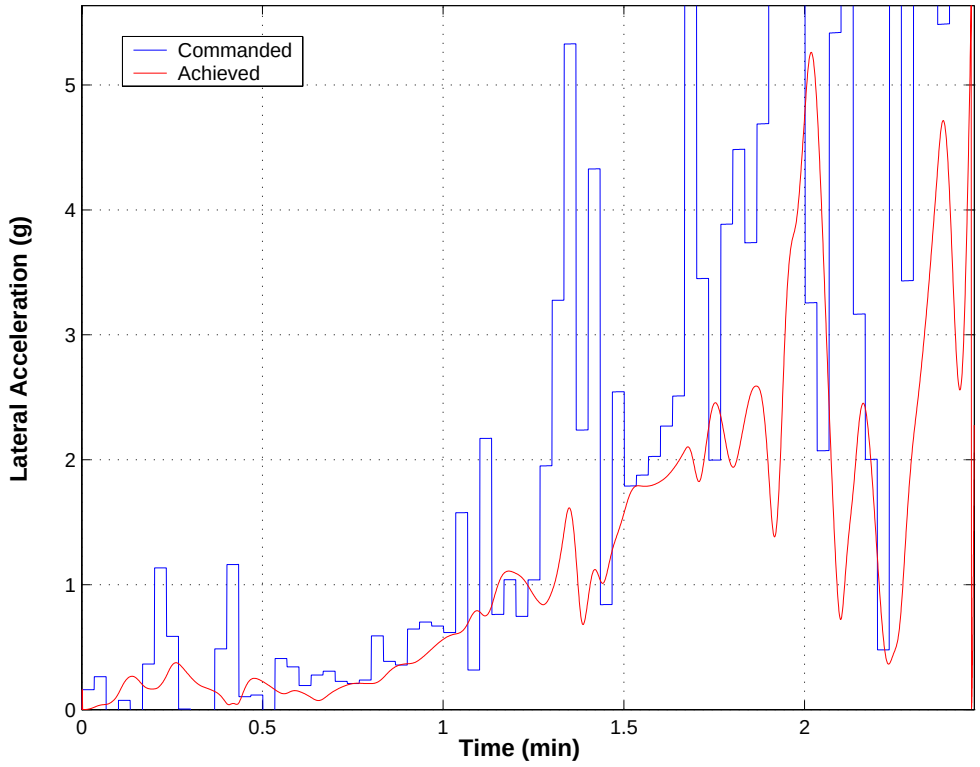


Figure II-8 Maneuver of interceptor toward the missile. The red line represents the achieved acceleration by the guidance system.

C. CHAFF CLOUDS

Chaff is the oldest, but still the most widely used, passive jamming technique against radar [18], [19]. The effect of this passive EA is the creation of a camouflaged background to cover the target.

A chaff cloud can be dispensed in many ways: a) by creating a corridor by continuous chaff dropping (moved by the effect of wind); b) by dispensing chaff bundles to simulate false targets; c) by dispensing chaff bursts to conceal the target; d) by dispensing chaff with forward-fired rockets to produce a chaff cloud in front of the target.

1. The RCS of the Chaff Cloud

The RCS of the chaff cloud is a key aspect of this method, since the main idea is to create an RCS that is equal to or greater than the target missile.

The determination of the chaff cloud RCS can be approximated via [20]: a) an aerodynamic solution that defines the cloud parameters as a function of space and time; or b) an electromagnetic solution of the scattering from this cloud. Another method of measuring the RCS is given by [20], by which integration is used to calculate the total power returned from the cloud relative to the power returned by a point target of a known RCS. In [19], the chaff cloud containing individual dipoles within the radar resolution cell is replaced by a single equivalent scatterer that returns the same amount of power to the radar. The size of the equivalent scatterer should then be equal to the volume of the resolution cell. The equivalent RCS is found as

$$\sigma_c = \pi\eta\theta_A\phi_E R^2\tau c/16 \ \ \mathrm{m}^2 \tag{2.2.1}$$

where η is the back-scattering coefficient in $\mathrm{m}^2/\mathrm{m}^3$, θ_A and ϕ_E are the azimuth and the elevation beamwidth in rad, respectively. R is the range in m to the resolution cell, τ is the radar pulsewidth in s, and c is the speed of light in $\mathrm{m/s}$.

Chaff is composed of a large number of shorted antenna dipoles [19], which can be made from paper, glass, fiber, or Capron, covered with a conductive layer (e.g., metallic foil) [18]. The length of the dipoles should be approximately equal to $0.46\lambda - 0.48\lambda$, where λ is the wavelength of the radar. The individual chaff orientation in the air is a matter of randomness. An approximate relationship that can be used to calculate the RCS of a randomly oriented dipole is given in [19] as

$$\bar{\sigma} = (1+\cos 2k\ell)^2.\frac{G^2(\theta,k\ell)\lambda^2}{\pi} \ \ \mathrm{m}^2 \tag{2.2.2}$$

where $k = 2\pi\lambda$, ℓ is the length of the dipole, and $G(\theta,k\ell)$ is the gain of the dipole. The gain of the dipoles varies with angle and orientation. Further analysis of (2.2.2) gives us the average RCS of the individual randomly oriented dipoles as $\sigma_1 = 0.15\lambda^2 \ \mathrm{m}^2$, whose length is given above [21].

2. Consideration of Chaff Tactics to Be Used

There are many ways to use chaff to degrade the radar performance [22], but most of them are not useful for our case.

a. False Target

Dispensing a bundle of chaff at various time intervals is one techniques: it causes the defense to exhaust their supply of interceptors. But this method is not useful if the real target is moving quickly. In this case, the discrimination of the target can be fast, since the target moves away from the cloud quickly. For example, the X-band Doppler radar integrates 20 pulses using a PRF of 150 Hz. This takes approximately 0.13 s, which is less than the usual update time of the fusion box.

b. Screening

Screening dispenses the chaff in order to create a corridor to submerge the target return. This method also is not useful in our case due to the requirement for the target to stay within the cloud. Creating a corridor that completely covers the target trajectory for a limited time is nearly impossible. For example, at time t = 85 s, the target has a velocity of about 2.3748 km/s, and its altitude is approximately 45 km above sea level. At this speed, the length of chaff corridor required is 11 km, as in Figure II-9. The target cannot dispense the chaff because the cloud will always be located at the back of the target and thus will not screen the target.

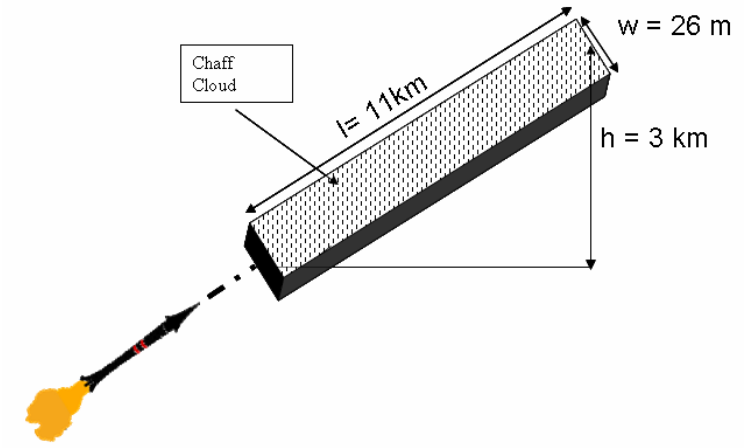


Figure II-9 The chaff corridor needed to cover the trajectory.

c. Forward-Fired Chaff

This technique uses a missile to launch a chaff rocket to the front of the target missile to create a chaff cloud. The launch time of the chaff rocket must be calcu-

lated precisely, so that the dispersion of the chaff takes effect and covers enough area to screen the target. The number of chaff dipoles needed for this action is found by using (2.2.3). To find the RCS, we first need to calculate the volume of the corridor.

Let us assume that the volume is a rectangular box with a length of 11 km and a width ten times larger than the radius of the target (e.g., 2.6 m). In this case, the surface area will be equal to 1,144,676 m^2. The RCS values of the target varies between 0.3 m^2 and 35 m^2 for different aspect angles. Therefore, within the time interval of 85-90 s, an RCS of 35 m^2 must be created to insure coverage of the target. The chaff cloud's RCS can be found from

$$\sigma_c = A_0(1 - e^{-n\sigma_1}) \tag{2.2.3}$$

where A_0 is the geometrically projected area of the cloud, *n* is the number of dipoles per unit of cloud projected area, and σ_1 is the average individual dipole RCS. When the dipoles become widely separated, this equation reduces to $\sigma_c = N\sigma_1$, where *N* is the number of dipoles contained in the cloud. The RCS value must be equal to the maximum RCS of the target to be covered. From here we find *N* = 259,260 dipoles within every range cell of the radar. In general, the RCS of the chaff should be at least twice as much as the platform that we intend to protect [18]. Furthermore, the position of the target and the radar cannot be known precisely in order to predict the look angle and the corresponding RCS value. The probability of a chaff RCS being greater than the target echo is addressed in [5]. That analysis can be applied to our case. For a 90 percent probability of our chaff cloud RCS being greater than the missile echo, the necessary number of dipoles required is N = 1,862,087 [5].

d. Obscuration

Obscuration is a method that degrades the radar performance by locating a chaff cloud at its look angle. The chaff cloud's first objective is to attenuate the radar's signal so that the radar maximum detection range of the target is degraded. The electromagnetic wave traveling through the chaff cloud is attenuated due to scattering and absorption from the individual dipoles in the cloud [21]. Figure II-10 illustrates this effect. The needed chaff amount for this method is at least 100 times greater than the self-

protection screening technique [22]. The incidence electric flux within the cloud due to the radar beam is given by [21]

$$S_i = P_t \frac{G_t(i)}{4\pi R^2} e^{-\gamma}$$
$$\gamma = \int_0^R \rho \langle \sigma_t(i) \rangle ds \tag{2.2.4}$$

where S_i is the incidence flux, $G_t(i)$ is the one-way antenna gain in the direction specified by the unit vector *i*, P_t is transmitted power, R is the distance from the radar and the volume elements in question, ρ is the density of the dipole and $\langle \sigma_t(i) \rangle$ is the average total scattering cross section of a dipole.

As in [18], we assume that the scattered radiation is isotropic and related to the average RCS of the cloud. The diffused fields through the chaff cloud are non-coherent, and the total field is the sum of the field caused by the individual field.

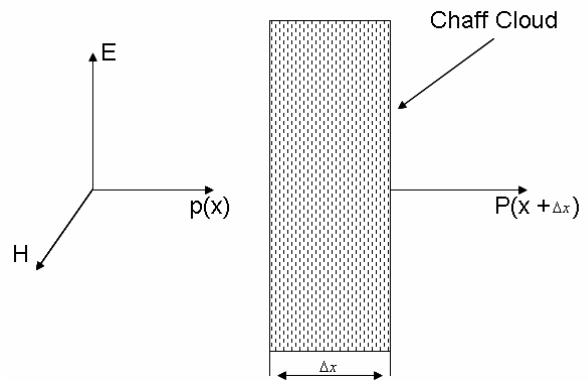


Figure II-10 Attenuation of the electric field within the chaff cloud (After [18]).

From the above assumption, the degree of attenuation of the power density in a chaff corridor of length Δx is given as [18]

$$\frac{dp}{p} = -\sigma_v dx$$
$$p = p_0 e^{-\sigma_v \Delta x} \tag{2.2.5}$$

where σ_v is the average RCS density of the chaff cloud, and p_0 is initial boundary condition for the incident wave. From (2.2.5), the degree of attenuation of the power density in decibels can be found as

$$p = p_0 10^{-0.1\alpha\Delta x} \tag{2.2.6}$$

where $\alpha = 4.3\sigma_v$ dB/m . The radar attenuation in range is found by using (2.2.6), with Δx increased by a factor two. Figure II-11 shows the geometry of the obscuration scenario.

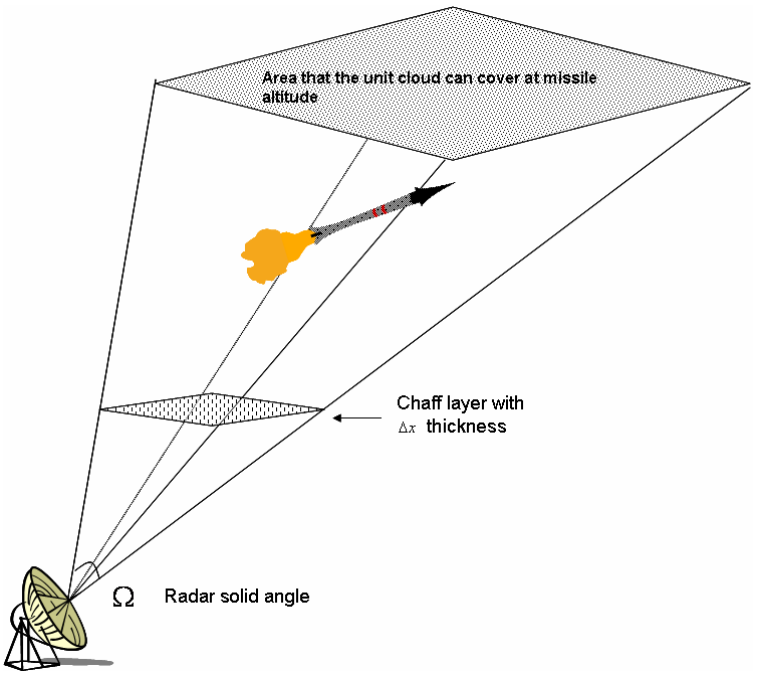


Figure II-11 Representation of the obscuration scenario.

To find out where the degradation factor in the radar range is a factor of 10, the radar equation can be used. With $\Delta x = 5$ m, the required chaff density will be $\sigma_v = 0.9302$ m^2/m^3 . The area required to be covered is 11 km $\times$1 km at an altitude of 45 km during the missile's time of flight. Thus, we need to cover a 2400 m $\times$222 m area on the line of sight of the radar at an altitude of 30,000 ft, which is a normal aircraft altitude. From the dimension of the field, the volume is calculated as 2,664,000 m^3. The required RCS for achieving a factor-of-10 degradation in the radar range is therefor

2.4781×10^6 m^2. To get this RCS value, the needed number of chaff dipoles is 1.8352×10^{10}. The weight of the chaff cloud can be found from [19]

$$\sigma_c = \frac{kW}{f} \tag{2.2.7}$$

where $k = 17000$ m^2-GHz/lb for aluminized glass chaff, which has a density of $2550\,\mathrm{kg/m^3}$, W is the weight in pounds, and f is the radar frequency in GHz. By using (2.2.7), the weight of the chaff bundle is found to be 489.67×10^3 kg, which cannot be carried by a single aircraft.

3. Effect of the Chaff Cloud

Of the chaff techniques discussed above, only the forward-fired chaff-rocket technique is feasible. To evaluate the effect of the chaff cloud created by a forward-fired rocket, the time until the target enters the chaff cloud is varied from 80 s to 147 s in the simulation. The exit time from the chaff cloud is a constant, 5-s interval after entrance into the cloud. While the target is within the cloud, the RF sensors cannot receive any position information. Due to the fusion center having received no new information within the 5-s window, the guidance commands for the interceptor cannot be updated, and the interceptor keeps on flying to the last received position of the target.

The average miss distance when not using chaff is 15 m. The resulting miss distance versus entrance time to the chaff cloud is shown in Figure II-12. With the corridor entrance at 80 s after the ICBM launch, the miss distance is small (the effect of the corridor tolerable), since the interceptor guidance can compensate by readjusting the flight path early in the scenario due to the relatively low speed of both missiles. At 80 to 85 s, the missile velocity is around 2.1 km/s. When the second staging of the ICBM occurs, the acceleration drops considerably. If this part of the scenario occurs within the chaff corridor (120 s), the effect is significant, since no updates are gathered by the radar to reflect this acceleration change. Consequently, the interceptor guidance commands remain constant and a large miss distance is incurred. The most significant effect of the chaff is realized when the target enters the corridor in the last few seconds of the interception. The effect may be enhanced further by making the chaff corridor length longer.

Figure II-13 shows the command lateral acceleration during the entire flight, with the target entering the corridor at 140 s (just prior to interception). Also shown is the achieved accelation. Note that the command lateral acceleration gets very small when the missile enters the corridor and increases quickly after the missile leaves the chaff corridor, in an attempt to compensate for the previous loss of target. The achieved guidance, however, is unable to compensate completely, since the corridor is placed so late in the flight. Consequently, a large miss distance is incurred.

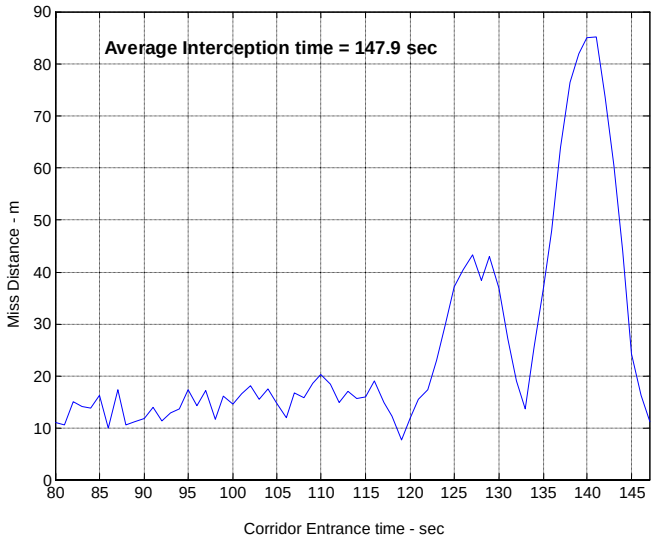


Figure II-12 Miss Distance versus target's entrance time to chaff corridor.

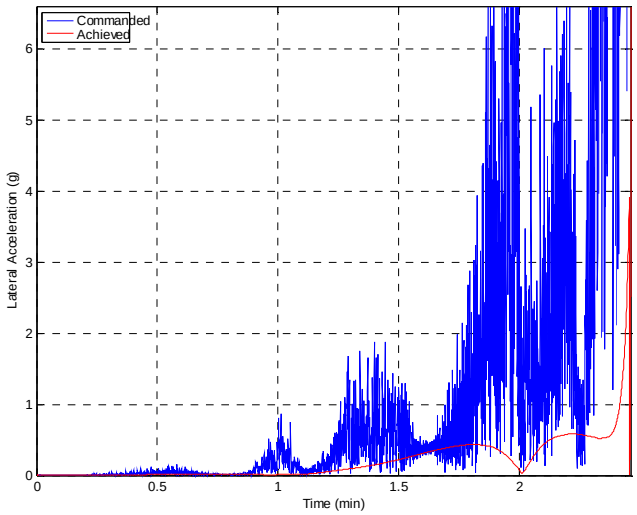


Figure II-13 Command lateral acceleration during flight. Entrance into chaff corridor at t =140 s.

D. ACTIVE JAMMING

In this section, we will discuss the effect of jamming by the ICBM on the RF sensors used in the boost phase defense system. All the ICBMs are assumed to be tracked if the radar systems are able to achieve a Line of Sight (LOS). All the radar systems are assumed to be located close enough to the ICBM launch positions.

1. Jamming Power Density

The power density returned from target ICBM must compete with the internal noise of the radar. The radar's parameters used in [5] are shown in Table II-3.

Table II-3 Generic Radar Parameters (After [5])

Parameter	Value
Frequency	10 GHz (X–Band)
Peak Power	1 MW
Antenna Gain	50 dB
Beamwidth	0.5×0.5 degrees
Pulsewidth	50 μs
PRF	150 Hz
Number of Pulses Integrated	20
Receiver Noise Factor	4

The internal noise can be calculated as in [7]:

$$N = kT_0 B_n F_n \tag{2.3.1}$$

where $k = 1.38 \times 10^{-23}$ J/deg is Boltzmann's constant, $T_0 = 290$ is standard temperature in degrees Kelvin, F_n is the noise factor of the radar, and $B_n = 1/\tau = 20$ kHz is the bandwidth of the radar. From this standpoint, the internal noise for the generic radar is $3.2 \times 10^{-16} = -154.9$ dBW . Most radar systems require receiving at least 10 to 20 dB SNR to detect a target [7], [23]; hence the jammer must introduce sufficient power into the radar antenna to eliminate the detection. The jamming power density at the radar receiver is given as

$$J_0 = \left(\frac{P_j G_j}{B_j} \right) \times \left(\frac{A_r}{4\pi R^2} \right) \tag{2.3.2}$$

where $P_j, G_j,$ and B_j are the jammer power, antenna gain toward the intended radar, and the noise bandwidth, respectively. Here A_r is the antenna aperture area, and R is the range between jammer and the radar. It is assumed that radar antenna is pointed directly to target and the all other losses are neglected [23]. The aperture of the generic X-band radar was not defined but can be found easily from the gain formula, if the aperture efficiency is assumed to be unity. The gain of the radar can be found from

$$G_r = \frac{41253}{\theta_A \phi_E} \tag{2.3.3}$$

where θ_A and ϕ_E are the azimuth and the elevation beamwidth in degrees, respectrively [24]. The other gain equation is

$$G_r = \frac{4\pi \rho A_r}{\lambda^2} \tag{2.3.4}$$

where ρ is the aperture efficiency assumed to be unity, and λ is the wavelength of the intended radar. Substituting (2.3.3) into (2.3.4), we find the radar aperture is $A_r = 11.82$ m.

When 10 W of jamming power are used to generate the noise jamming, the resulting power is far above the noise level of the radar. The jammer antenna gain is assumed to be 10 dB. In [23], it is implied that "the 10-dB antenna gain could be obtained with a simple cavity antenna, flush-mounted on the surface of the final stage and covering a sector $90°$ in azimuth by $45°$ in elevation. This tactic would obviate the necessity of knowing the radar's location."

Even in the worst case, in which the EA system is a broadband noise jammer that has no information about the radar parameters, the resulting jamming-to-noise ratio is on the order of +70 dB, which prevents any target detections. The jamming power density at the radar's antenna versus the range from the jammer to the radar is shown in Figure II-14 through Figure II-16.

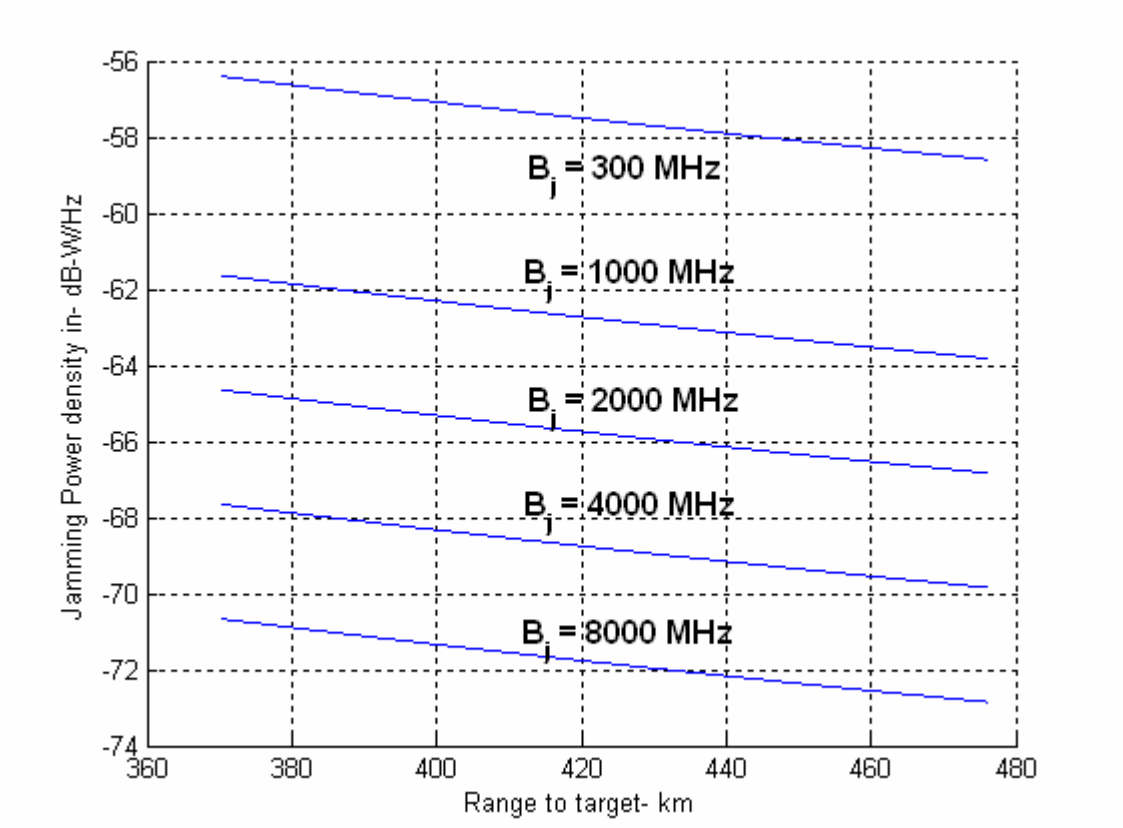


Figure II-14 Jamming power density at the RF-1 radar antenna versus range from jammer to target. Jammer power is assumed to be 10 watts.

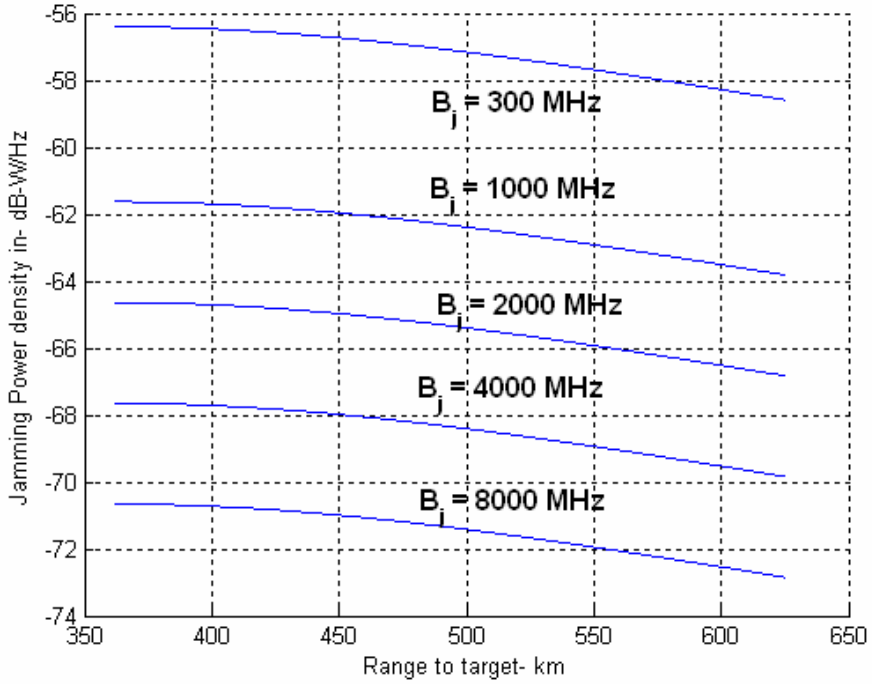


Figure II-15 Jamming power density at the RF-2 radar antenna versus range from jammer to target. Jammer power is assumed to be 10 watts.

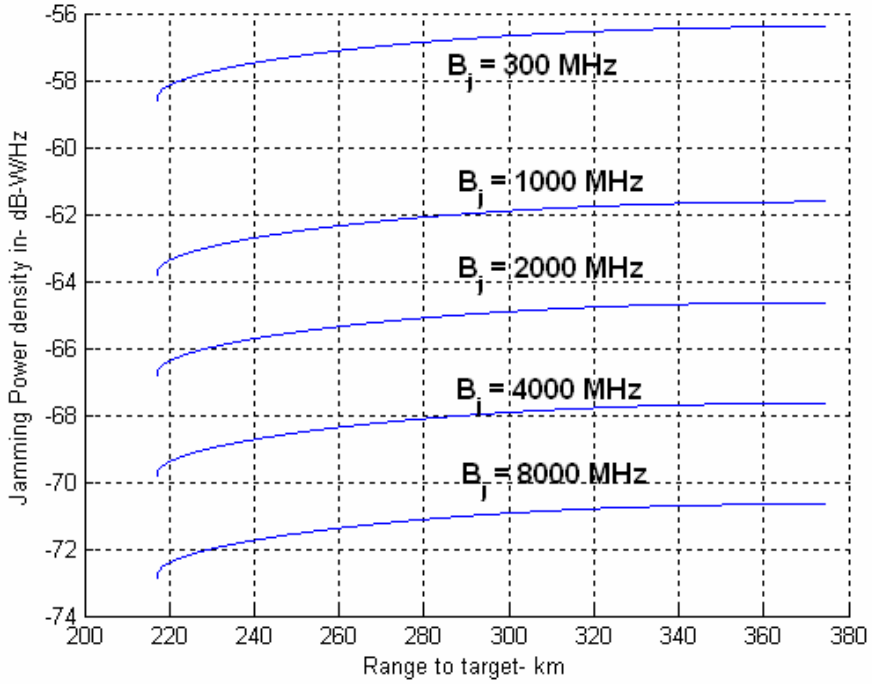


Figure II-16 Jamming power density at the RF-3 radar antenna versus range from jammer to target. Jammer power is assumed to be 10 watts.

2. Jamming Effects

The effect of the jamming is to force the radar to use angle-only measurements for tracking the target's position. When the jammer power is such that the radar can achieve detections, the low Signal-to-Jamming Ratio (SJR), shown in Figure II-17, causes the standard deviation of the range and angle measurement error to be large, which gives large position errors. If the fusion center does not use an advanced algorithm, but either the weighted average or just the average, the guidance produced by the navigation algorithm may either force the missile to crash or force the fusion center to increase the data update interval to smooth the position error.

The simulation runs for a jammer power of 10 W and a bandwidth of 4 GHz (X-band only). Even if less power is used, the average miss distance increases by 7 m. The resulting tracking range error for each RF sensor is shown in Figure II-18; the angle error for each RF sensor is shown in Figure II-19.

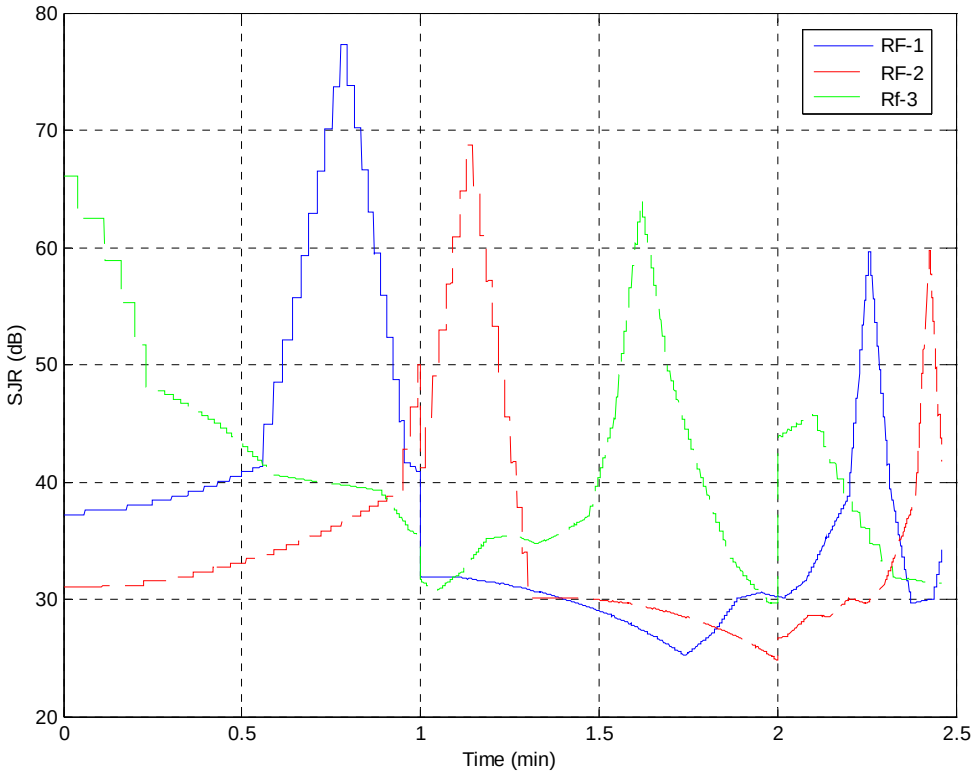


Figure II-17 Signal to Jam Ratio. P_j = 10 W, B_j= 4 GHz

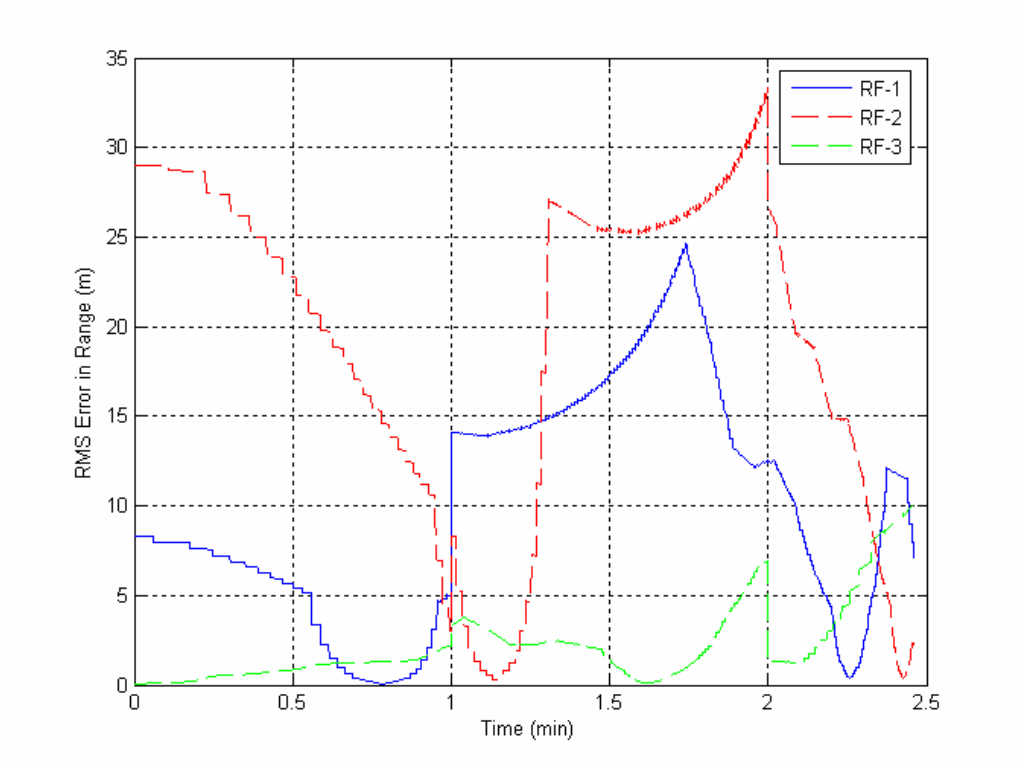


Figure II-18 RMS errors in range introduced while target is jamming. P_j = 10 W, B_j= 4 GHz.

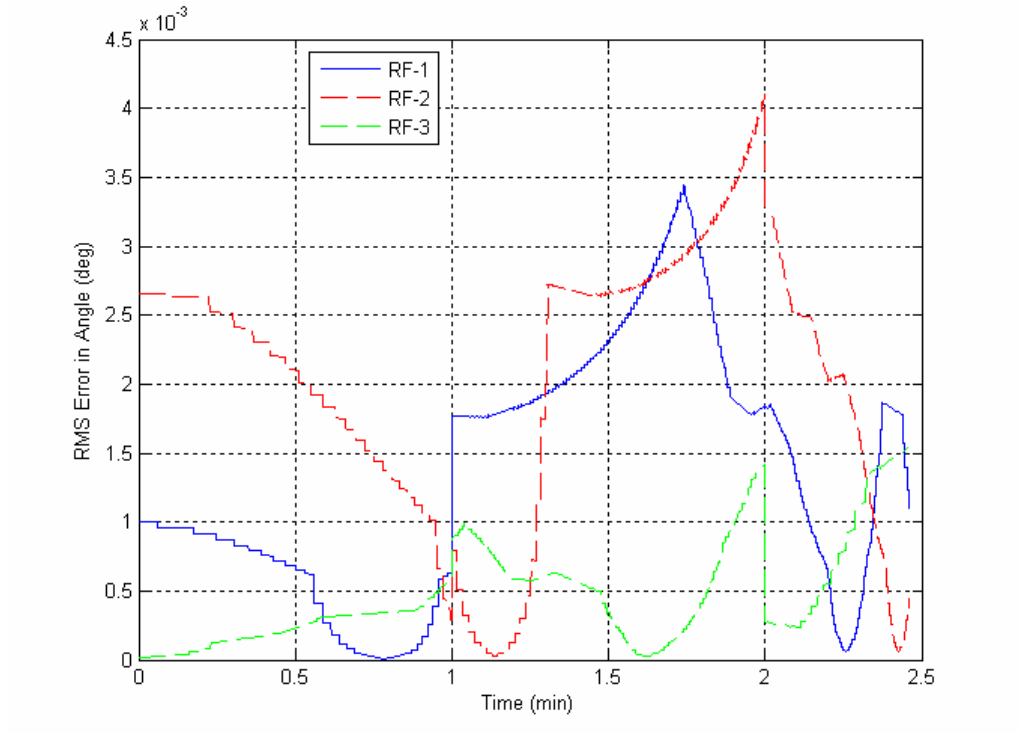


Figure II-19 RMS errors in angle introduced while target is jamming. P_j = 10 W, B_j= 4 GHz.

The average power of the jammer may also be increased. Figure II-20 taken from [22], implies that with an extra weight of 100 lbs, a 10-kW narrowband HPM can be used as a jammer emitter.

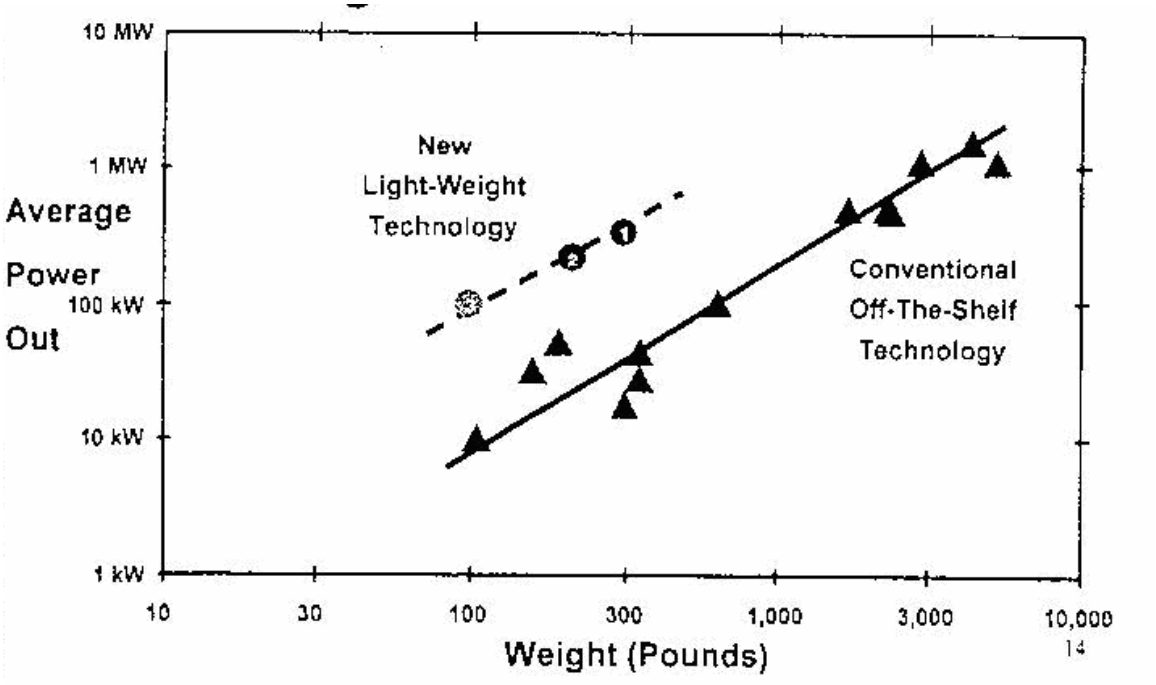


Figure II-20 Average power versus weight of the narrowband HPM (From [22]).

If we increase the jammer power to 10 kW while keeping the bandwidth the same (e.g., 4 GHz), the resulting SJR is shown in Figure II-21, the RMS error in range is shown in Figure II-22, and the RMS error in angle is shown in Figure II-23. The miss distance is immense. The interceptor cannot finish its mission; it has an average miss distance of 10,000 m due to the large uneven commanded acceleration.

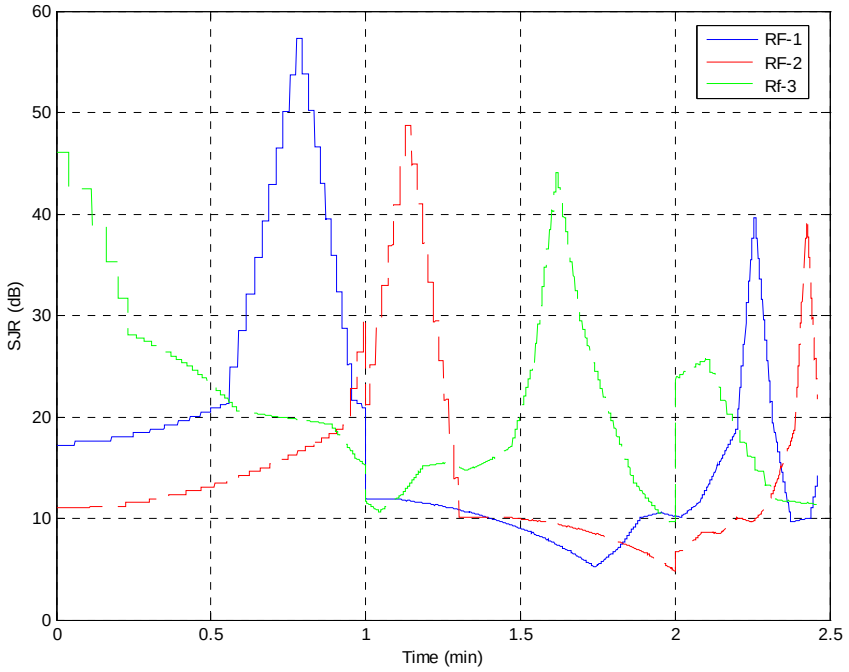


Figure II-21 Signal to Jam Ratio: P_j = 10 kW, B_j= 4 GHz

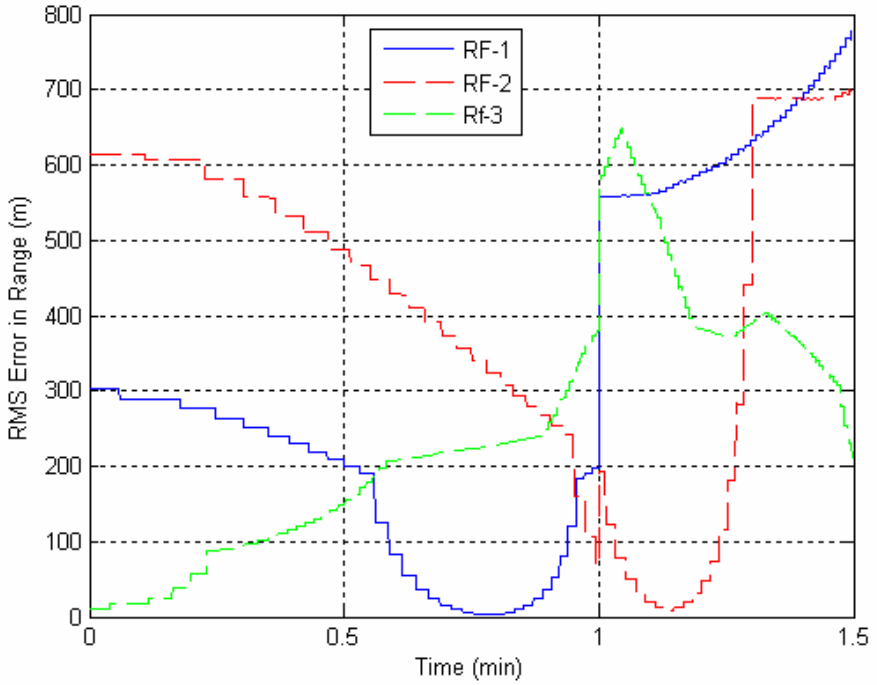


Figure II-22 RMS errors in range introduced while target is jamming: P_j = 10 kW, B_j = 4 GHz.

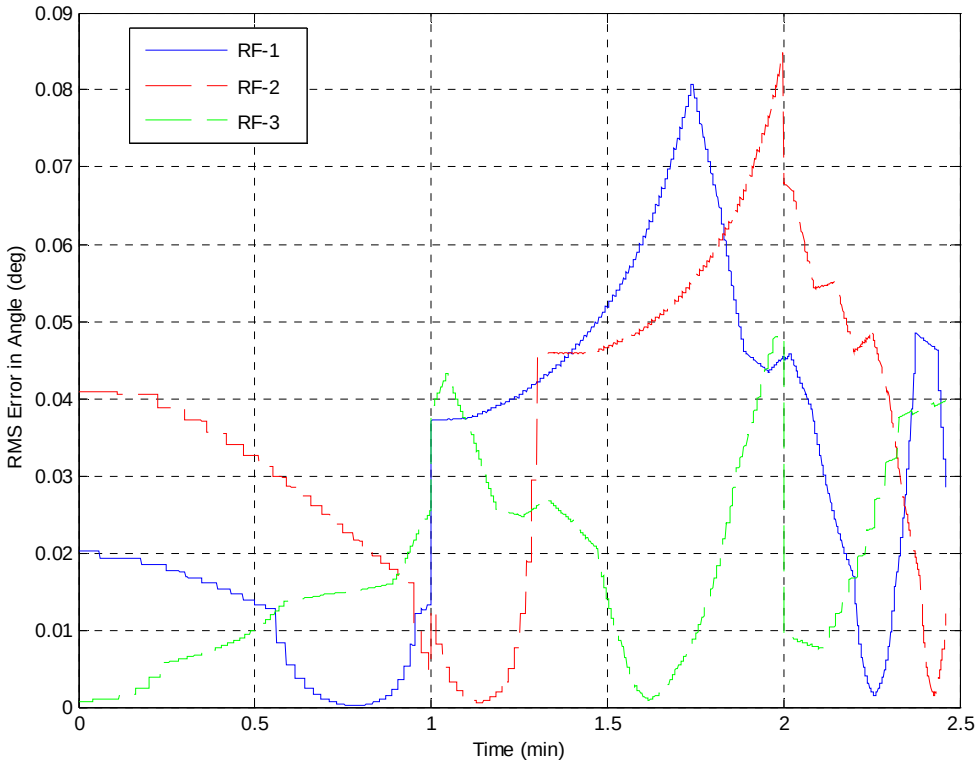


Figure II-23 RMS errors in angle introduced while target is jamming: P_j = 10 kW, B_j = 4 GHz.

If the jamming denies the range detection and forces the radar to use angle-only measurement, the tracking position error is even worse. The position error using triangulation (use of two or more RF sensor's data to decide the position) is shown in Figure II-24. The large commanded lateral acceleration that causes the interceptor to crash is shown in Figure II-25. In order to complete the flight, the data update time is increased to 5 s. The resulting miss distance is on the order of 100 km.

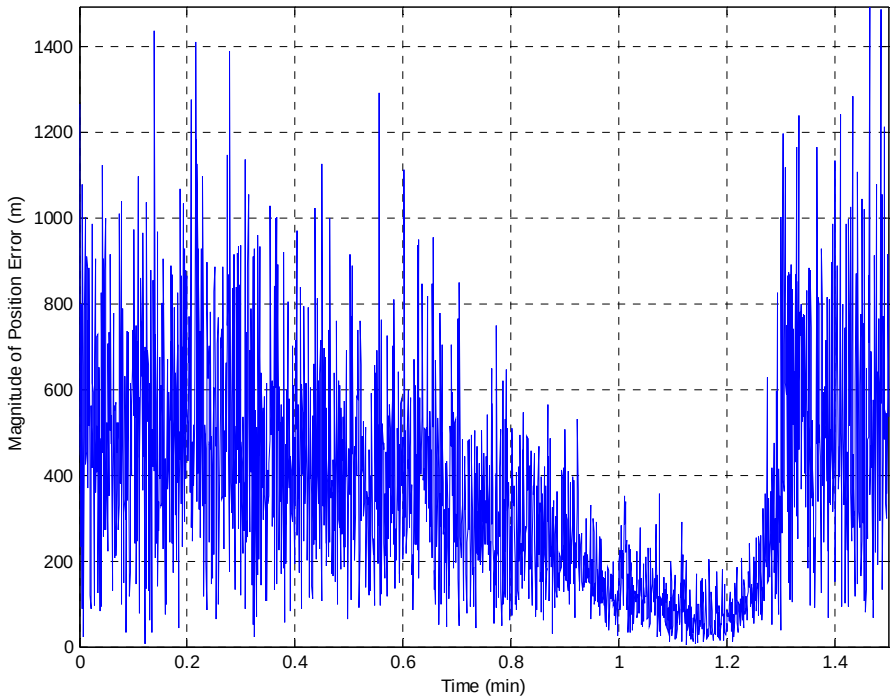


Figure II-24 Position error introduced using triangulation. The resulting miss distance is 539 km.

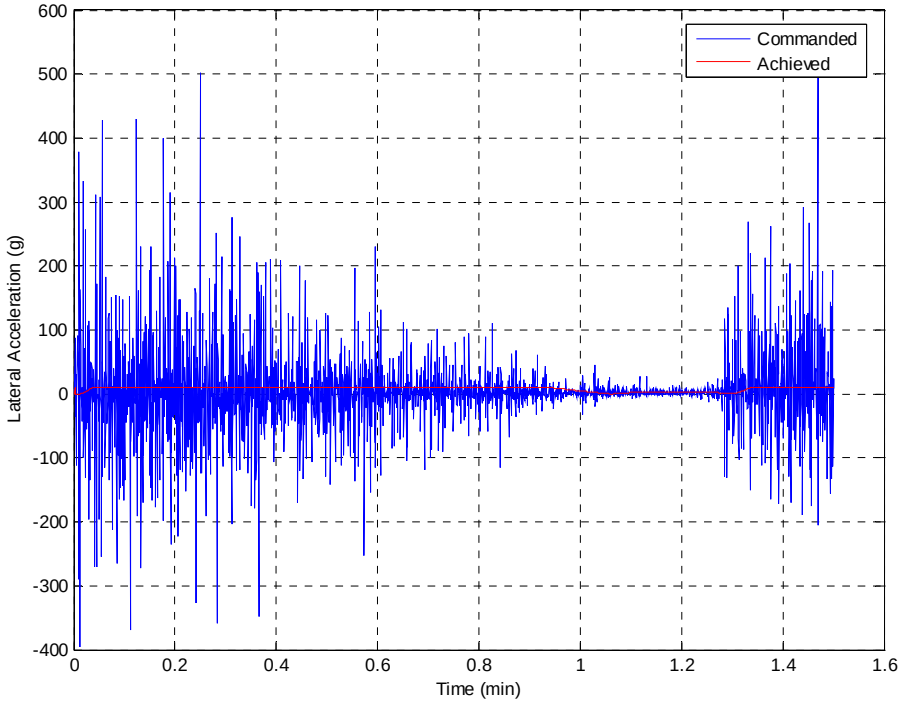


Figure II-25 Lateral acceleration of the interceptor while using a fusion center that uses the triangulation.

E. ACTIVE DECOYS

In this section, the effect of active decoys will be investigated.

The monopulse radar has a natural defense against EAs. Modern radar systems use advanced radar techniques (e.g., monopulse tracking), and efficient electronic protection (EP) to reduce the significant effect of EAs. We can consider a monopulse radar operating in frequency agility mode and exploiting Anti Range Gate Stealing (ARGS) EP techniques at presently difficult threats to be jammed [25]. There should be an effective system to defeat the monopulse radars, and using active decoys is one of them.

There are two types of decoys in use. The first is a towed decoy, which is composed of a small flying repeater jammer at the back of the airborne target, that mimics the signature of the real target to lure the threat missile onto itself. Since there is no appropriate place to hang the decoy, this type is not suitable for ballistic missile defense. Second is the expendable decoys, which are dispensed to defeat the incoming missile. The expendable decoys are described below.

1. Repeater Decoys

To apply the seduction, the decoy repeater needs to radiate enough power into the radar receiver. The implementation does not pose any problem. One of the antennas receives the victim radar signals. After processing this signal, the other antenna re-radiates it. A typical repeater block diagram is shown in Figure II-26.

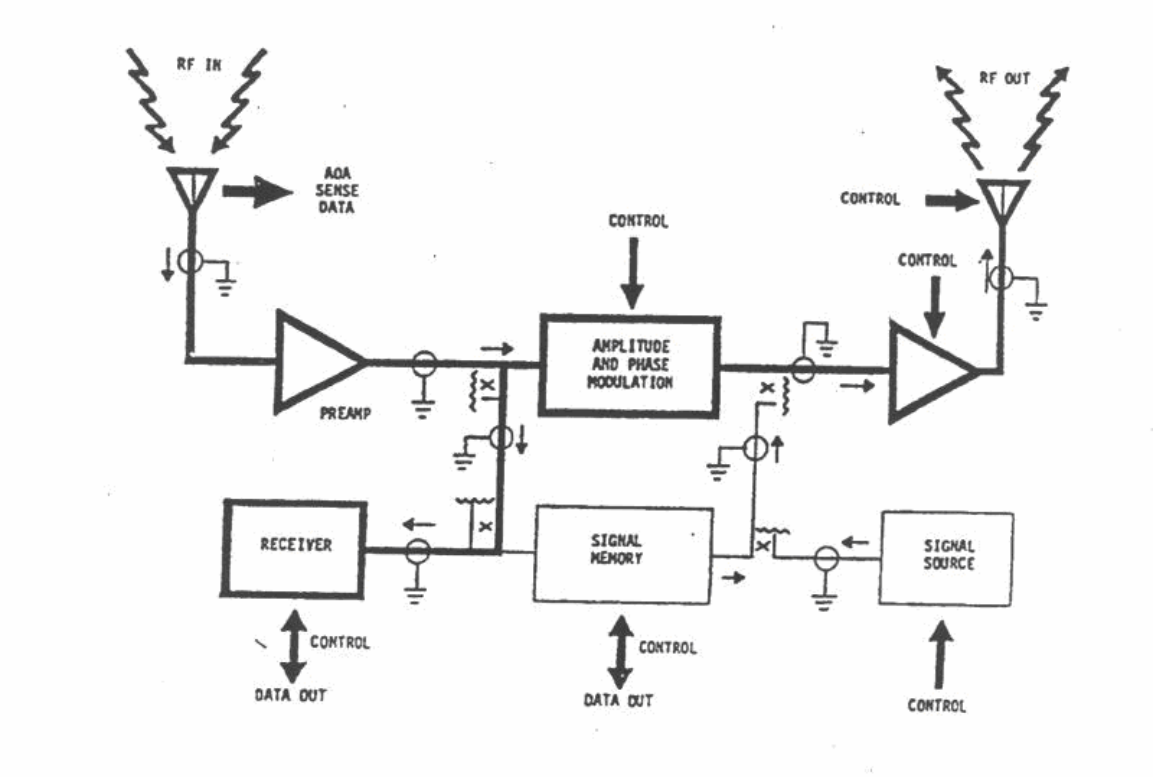


Figure II-26 Typical repeater block diagram (From [26])

The repeater performance can be defined in two ways: (a) the jam-to-signal ratio (JSR) and (b) the repeater gain, which is a constant. In [26], JSR is given as

$$\frac{J}{S} = \frac{G_{dr} G_{dt} G_d \lambda^2}{4\pi\sigma L_p^2} \tag{2.3.5}$$

where G_{dr} is the repeater receiver antenna gain, G_{dt} is the repeater transmitter antenna gain, G_d is the repeater amplifier gain, L_p is the polarization loss, λ is the radar wave-length, and σ is the target RCS. As is seen from (2.3.5), the ratio is range independent. When operating in saturation, (2.3.5) becomes range dependent as given by

$$\frac{J}{S} = \frac{4\pi R^2 P_{\max}}{P_t G_t \sigma L_p} \tag{2.3.6}$$

where R is the range to the missile, $P_{\max}$ is the maximum repeater power output, P_t is the radar transmitted power, and G_t is the radar antenna gain. Here the polarization loss is an

important issue. Because the decoys designers cannot be sure about the real polarization of the radar being attacked a circular or $45°$ slant polarization may generally be used. This causes a 3 dB loss.

If the distance between the decoy and the dispensing platform is much less than the distance between the platform and the threat missile, and the repeater receiver and transmitting antenna are assumed to be identical in gain, the gain of the repeater is given as [18]

$$G_d = \frac{K_j \sigma}{G_{dr}^2} \tag{2.3.7}$$

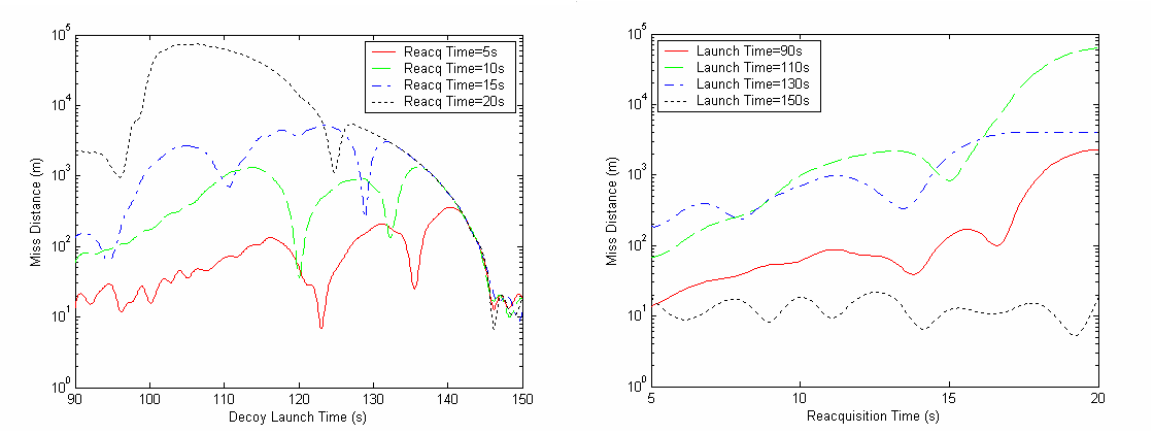
where K_j is defined as the SJR.

2. Passive Reflectors

Some good reflector shapes can also produce high RCS. The reflector must have the smallest possible external dimensions and have broad reflection patterns, since the orientation of the reflector cannot be known beforehand [18]. Corner reflector is the kind of device that has a large RCS compared with its physical area. Other types of reflectors can be produced via Lunenburg lenses and Van Atta arrays. The calculation for various types of passive reflectors are given in [18].

3. Effect of Decoys

In the simulation, the decoys are not modeled in detail. Only the RCS of the decoys at any given time is given, so the missile tracking system is lured toward the decoys. Because the decoys do not have any propulsion mechanism to move, whenever they are released from the ICBM, the speed of the decoys is the same as the velocity of the ICBM initially; but it starts to decrease due to gravitation and the drag of the air. The separation of the target and the decoy is so smooth that the initial phase can be cumbersome for the radars. A good analysis of the track transfer to the decoy is given in [5]. In Figure II-27 (a), the decoy release time versus miss distance is shown. In Figure II-27 (b), reacquisition of the ballistic missile time of radar versus miss distance is shown.



(a) (b)

Figure II-27 Miss Distance as a function of decoy (a) release time and (b) reacquisition time (From [5]).

As shown in Figure II-27, when the decoy's release time approaches the time-to-go, interception time, the miss distance will also increase. The most important point illustrated in the figure is the reacquisition time. If the reacquisition time is large enough, the missile interceptor cannot recover. Monopulse radar can discriminate two targets if their separation distance is more than the length of a resolution cell, which is 3.5 km if the radar-to-target distance is assumed to be 400 km. Because the target and decoy separation is smooth, the 3.5-km distance can be reached in a short time. Even if the speed of the ICBM can be varied depending on the decoy release time, the discrimination is done very quickly.

A representation of the radar resolution cell is shown in Figure II-28 in which θ_A is the azimuth beamwidth in rad, ϕ_E is the elevation beamwidth in rad, R is the range in m, c is the speed of light in m/s, and τ is the pulse width in s. The discrimination depends on the azimuth and elevation beamwidth and the range to the related cell.

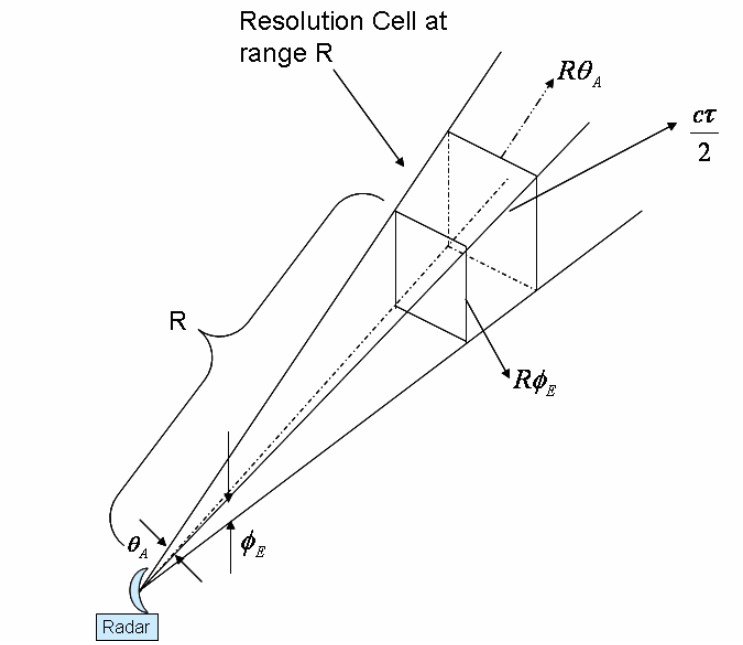
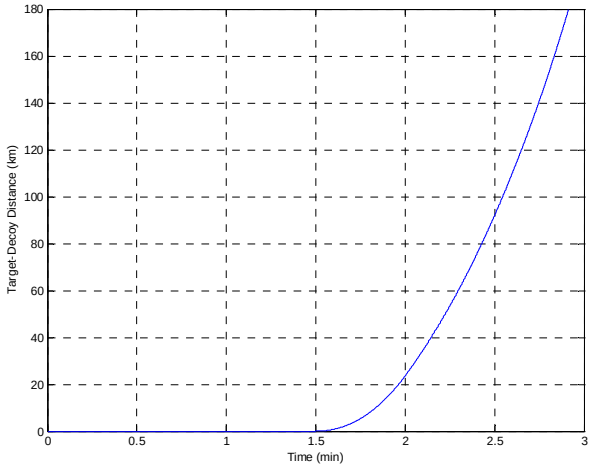
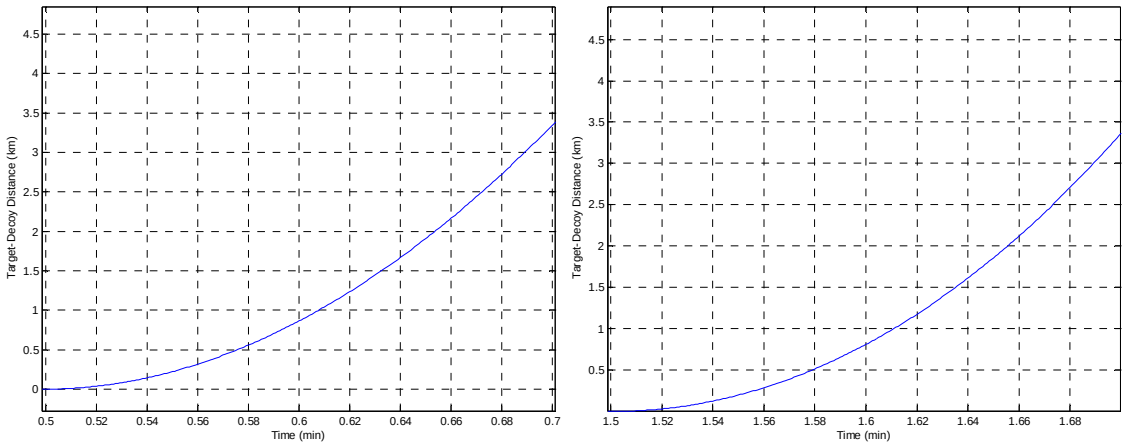


Figure II-28 Radar resolution cell

Figure II-29 shows the ICBM-decoy separation for various release times. When the decoys are released at t = 90 s, the separation takes palce within 10 s. Even when the decoy is released at t = 30 s, as is shown in Figure II-29 (b), the 3.5 km is reached within 11 s. That means that the reacquisition time cannot be greater than 12 s. Due to this discrimination, we do not expect the miss distance to increase more than 100 m.



(a)



(b) (c)

Figure II-29 Target-decoy separation when the decoy is released at time (a) *t* = 90 s and (b) *t* = 30 s, and (c) *t* =90 s.

F. SUMMARY

This chapter investigated the EAs that can be used against RF sensors during the ICBM boost-phase defense system. The study included the reduction in the RCS of the target missile, the chaff tactics that can be used, the active barrage noise, and the use of passive reflectors and/or active decoys.

All countermeasures are effective over a poorly designed fusion center in which a weighted average of the sensed position from each of the radars is calculated. The advanced fusion algorithm (i.e., using the Bayesian approach and/or Kalman filtering) is expected to reduce the effect of EAs. This counteraction is the subject of the next chapter.

THIS PAGE INTENTIONALLY LEFT BLANK

III. ADVANCED FUSION TECHNIQUES FOR SINGLE TARGET

This chapter presents an advanced fusion algorithm, using a Kalman filter and Bayesian analysis, to prevent or minimize the effect of the considered EAs. In the first section, we design a Kalman filter by using the least squares estimation method for the fusion center. In the second section, we will the previously developed Bayesian fusion [6] into the simulation.

A. KALMAN FILTERING

In this section, we will define the system model and the algorithm that we applied to our simulation, by using the definition [27]:

> An optimal estimator is a computational algorithm that processes measurements to deduce a minimum error estimate of the state of a system by utilizing; knowledge of system and measurement dynamics, assumed statistics of system noises and measurement errors, and initial condition information.

1. System Model

The fundamental system model that was used in the 3-D simulation of the ballistic missile intercept is given in [5]. The gravity field was calculated by using a perfectly round earth model with

$$g = \frac{GM}{r^2} \tag{3.1.1}$$

where $G = 6.67 \times 10^{-11}\ \mathrm{m}^3/(\mathrm{kg.s}^2)$ is the gravitational constant, $M = 5.98 \times 10^{24}$ kg is the mass of Earth, and r is the distance from the center of the Earth to the point that is the magnitude of the position vector in m, based on the assumption that the Earth is a nonrotating, perfect sphere with uniform density [28].

The net forces acting on the missile are explaned as

$$\mathbf{F}_{\mathbf{net}} = \mathbf{T} + \mathbf{W} \tag{3.1.2}$$

where $\mathbf{T}$ is the trust vector, which is in the direction of the velocity unit vector, $\vec{v}$ and $\mathbf{W}$ is the weight vector, which is in the direction toward the Earth center, but in the oppo

site direction of the unit vector of the position. Before giving the magnitude of these two major forces, we consider the Drag vector, $\mathbf{D}$, which is assumed to be zero over the missile body.

Unless otherwise stated, the missile has a perfect aerodynamic body that reduces the drag; however, the drag vector cannot be neglected at altitudes less than 300,000 ft, which is approximately 100 km [29]. Because most of the boost-phase missile interception takes place below this altitude, the drag in the opposite direction of the velocity unit vector needs to be accounted for as a major force that affects the missile body. Using this consideration, the net force vector can be redefined as:

$$\begin{aligned} \mathbf{F}_{\mathbf{net}} &= \mathbf{T} + \mathbf{W} + \mathbf{D} \\ \mathbf{T} &= -\frac{dm}{dt} g I_{sp} \\ \mathbf{W} &= mg \\ \mathbf{D} &= \frac{\rho V^2 A C_d}{2} \end{aligned} \tag{3.1.3}$$

where dm/dt is the in-stage fuel consumption in $\mathrm{kg/s}$, g is the gravitation that is given in (3.1.1), I_{sp} is the specific impulse in s, m is the mass of the missile in kg, ρ is the air density [30], A is the cross-sectional area of the missile body, and C_d is the zero-lift drag coefficient.

The variant acceleration motion model is given in (3.1.3). The position and the velocity change in time can be written as

$$\begin{aligned} \mathbf{x}(k+1) &= \mathbf{x}(k) + \mathbf{V}(k)\Delta \\ \mathbf{V}(k+1) &= \mathbf{V}(k) + \mathbf{a}(k)\Delta \\ \mathbf{a}(k+1) &= \frac{\mathbf{F}_{net}(k)}{m(k)} \end{aligned} \tag{3.1.4}$$

where k is the time index of the discrete time, Δ is the time interval, $\mathbf{x}$ is the position, $\mathbf{V}$ is the velocity, and $\mathbf{a}$ is the acceleration vector, respectively.

2. Dynamic System Matrices

All vectors have three dimensions that can be represented in a three-axis coordinate system. The state-vector of the dynamic model in six-dimensions is

$$\mathbf{X}(k) = \begin{bmatrix} x & y & z & V_x & V_y & V_z \end{bmatrix}^T \tag{3.1.5}$$

where the first three variables define the position of the missile, and the last three define the velocity of the missile in the Cartesian coordinate system. The missile dynamics may be represented by the vector-matrix equation as

$$\mathbf{X}(k+1) = \mathbf{F}\mathbf{X}(k) + \mathbf{v}(k) \tag{3.1.6}$$

where $\mathbf{X}$ is the state vector defined in (3.1.5), $\mathbf{F}$ is the state transition matrix, and $\mathbf{v}$ is the plant (system) noise having a covariance of $\mathbf{Q}$, which accounts for any unmodeled missile acceleration that can result in a deviation from the intended trajectory.

The radar sensor can measure the target's range, azimuth, and elevation. The previous simulation assumed that all sensors convert the measured data to a ECEF Cartesian coordinate system independently and then send it to the fusion center. The fusion center is assumed to have all information in the same format (a Cartesian coordinate system). From this assumption, the measurement equation can be written as

$$\mathbf{Z}(k) = \mathbf{H}\mathbf{X}(k) + \boldsymbol{\omega}(k) \tag{3.1.7}$$

where $\mathbf{Z}(k)$ is measurement vector that can be defined as

$$\mathbf{Z}(k) = \begin{bmatrix} x_m \\ y_m \\ z_m \end{bmatrix} \tag{3.1.8}$$

$\mathbf{H}$ is the measurement matrix, which can be defined as

$$H = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix} \tag{3.1.9}$$

and $\boldsymbol{\omega}$ is the measurement (sensor) error, which has a covariance of $\mathbf{R}$.

The only unknown matrix, the transition matrix, can be found by substituting (3.1.3) into (3.1.4). The resulting transition matrix is as follows.

$$\mathbf{F}=\begin{bmatrix} 1 & 0 & 0 & \Delta & 0 & 0 \\ 0 & 1 & 0 & 0 & \Delta & 0 \\ 0 & 0 & 1 & 0 & 0 & \Delta \\ -W\Delta & 0 & 0 & 1+(T-D)\Delta & 0 & 0 \\ 0 & -W\Delta & 0 & 0 & 1+(T-D)\Delta & 0 \\ 0 & 0 & -W\Delta & 0 & 0 & 1+(T-D)\Delta \end{bmatrix} \tag{3.1.10}$$

where

$$W=\frac{GM}{|\mathbf{x}|^3} \tag{3.1.11}$$

$$T=\frac{\frac{dm}{dt}I_{sp}GM}{|\mathbf{x}|^2|\mathbf{v}|m} \tag{3.1.12}$$

$$D=\frac{\rho GMC_dA}{2m|\mathbf{x}|^2} \tag{3.1.13}$$

and, $|\mathbf{x}|$ and $|\mathbf{v}|$ are the magnitude of the position and the velocity vector, respectively. A detailed derivation can be found in [29].

3. Noise and Covariance

The noise terms defined above are assumed to be zero mean (implying an unbiased sensor) with the following covariance structure [31]:

$$\begin{aligned} \mathbf{Q}_k &= \mathrm{cov}(\mathbf{v}(k)) \\ \mathbf{R}_k &= \mathrm{cov}(\boldsymbol{\omega}(k)) \end{aligned} \tag{3.1.14}$$

where cov( ) is the covariance operator, and all noise sequences are uncorrelated as in [31]

$$\begin{aligned} &\mathrm{cov}(\mathbf{v}(j),\mathbf{v}(k))=\varnothing, \quad \forall j\neq k \\ &\mathrm{cov}(\boldsymbol{\omega}(j),\boldsymbol{\omega}(k))=\varnothing, \quad \forall j\neq k \\ &\mathrm{cov}(\mathbf{v}(j),\boldsymbol{\omega}(k))=\varnothing, \quad \forall j,k \end{aligned} \tag{3.1.15}$$

The sensor, which tracks the target, has known measurement noise depending on its accuracy. The sensitivity and the accuracy of the RF sensors are given in [5] in three dimensions (R,θ,ϕ):

$$\sigma_{\theta,\phi} = \frac{\theta_{3dB}}{K\sqrt{(2(S/N)_1)N_i}} \quad \text{(angular error)} \tag{3.1.16}$$

$$\sigma_R = \frac{c\tau}{2}\frac{1}{K\sqrt{(2(S/N)_1)N_i}} \quad \text{(range error)} \tag{3.1.17}$$

where θ_{3dB} is the beamwidth of the radar in the related angular dimension, K is the RMS angle error for monopulse radar and is assumed to be 1.7 for simulation, $(S/N)_1$ is the single pulse SNR of the radar, and N_i is the number of pulses that are integrated to increase the SNR. The standard deviation in angle (or bearing) is assumed to be the same for the azimuth, θ, and the elevation, ϕ, due to the same beamwidth used (a pencil beam).

The covariance of the measurement will be

$$\mathbf{R} = \begin{bmatrix} \sigma_R^2 & 0 & 0 \\ 0 & \sigma_\theta^2 & 0 \\ 0 & 0 & \sigma_\phi^2 \end{bmatrix} \tag{3.1.18}$$

Because the radar converts the polar measurement into Cartesian coordinates, the linear measurement becomes nonlinear. The nonlinear transformation to a Cartesian coordinate system is given by

$$\mathbf{z}(k) = \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} R\sin\theta\cos\phi \\ R\sin\theta\sin\phi \\ R\cos\theta \end{bmatrix} \tag{3.1.19}$$

using the conversion matrix [12]: where R is the radius of the sphere from the center, the zenith angle ϕ is measured from the positive z-axis and describes the conical surface with its apex at the origin; and the azimuth angle θ is the same as in the cylindrical coordinate system. With the nonlinear transformation, we must, now, approximate the covariance of the measurement in the new coordinate system.

Let $y = f(x)$, a nonlinear transformation of x. We know that for a small error, ε_x:

$$f(y) = f(x_0 + \varepsilon_x) \cong f(x_0) + f_x(x_0)\varepsilon_x \tag{3.1.20}$$

where this is the first order Taylor series expansion about x_0, and $f_x(x_0)$ is the gradient of the transformation evaluated at x_0. The mapping of the covariance can be formulated as [32]

$$\text{cov}(\varepsilon_y) = f_x(x_0)\,\text{cov}(\varepsilon_x)(f_x(x_0))^T \tag{3.1.21}$$

where superscript *T* is the matrix transpose, and ε_y is the errors in function $f(y)$.

By taking the gradient of the transformation evaluated at the measured values in (3.1.19) and applying the formula given in (3.1.21), we can obtain the mapped covariance matrix as

$$\mathbf{R}_{\begin{bmatrix} x \\ y \\ z \end{bmatrix}} = \begin{bmatrix} \sigma_R^4 \sin\theta_m \cos\phi_m & \sigma_R^2\sigma_\theta^2 \cos\theta_m \cos\phi_m & -\sigma_R^2\sigma_\phi^2 R \sin\theta_m \sin\phi_m \\ \sigma_\theta^2\sigma_R^2 \sin\theta_m \sin\phi_m & \sigma_\theta^4 R \cos\theta_m \sin\phi_m & \sigma_\theta^2\sigma_\phi^2 R \sin\theta_m \cos\phi_m \\ \sigma_\phi^2\sigma_R^2 \cos\theta_m & -\sigma_\phi^2\sigma_\theta^2 R \sin\theta_m & 0 \end{bmatrix} \tag{3.1.22}$$

where subscript *m* stands for measured value at time *k*.

The process (plant) noise must be modeled to account for the unmodeled or arbitrary inputs that may cause the vehicle to stray from a deterministic trajectory. Using the target acceleration to model the random target motion is suggested in [33]. The standard acceleration given in [33] is a first-order Markov process, which can be written as:

$$a(k+1) = \rho_m a(k) + \sqrt{1-\rho_m^2}\,\sigma_m r(k) \tag{3.1.23}$$

where the maneuver correlation coefficient, ρ_m, is given in terms of sampling time, Δ, and maneuver time constant, τ_m, as

$$\rho_m = e^{-\frac{\Delta}{\tau_m}} \tag{3.1.24}$$

The σ_m is the maneuver standard deviation, and $r(k)$ is a zero-mean Gaussian variable with unit standard deviation [33]. The target acceleration is shown in Figure III-1.

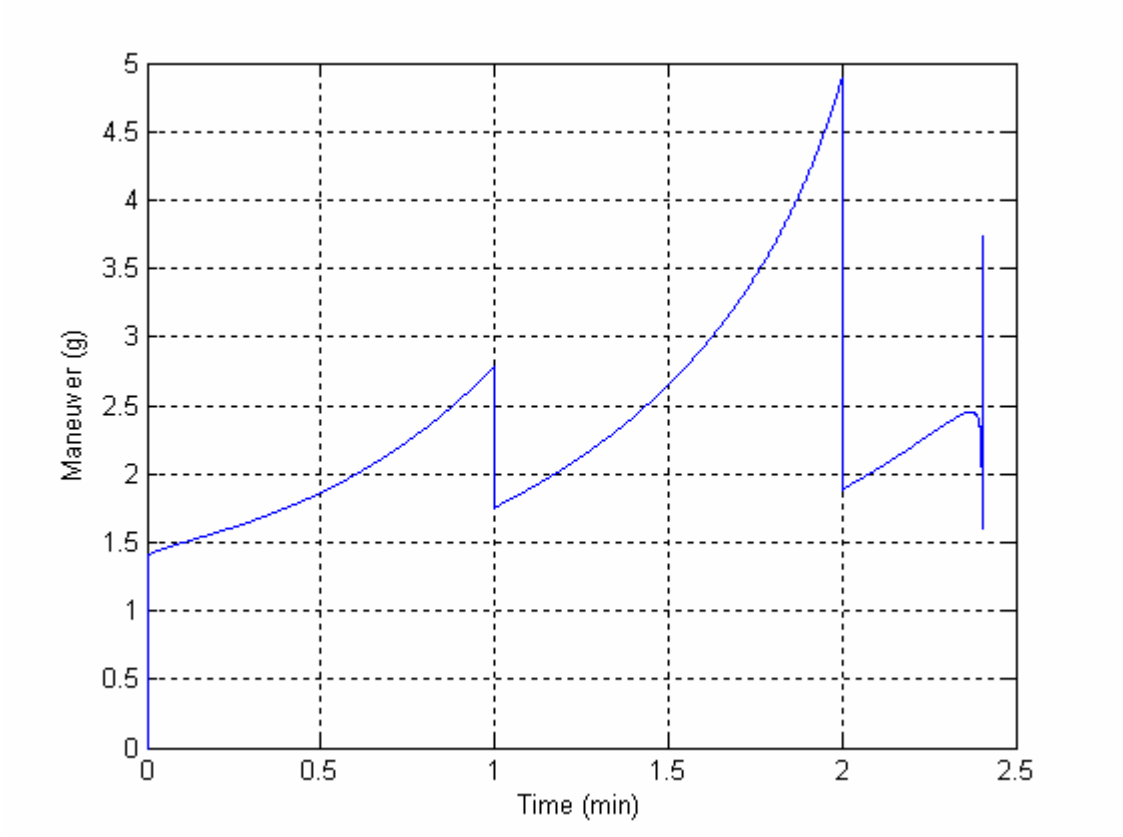


Figure III-1 Target maneuver (acceleration) during boost phase

The maximum of the acceleration is 4.89g. For every stage change, the acceleration drops to an initial value and starts to increase again. The distribution of the acceleration is exponential within the stage. Using one third of the maximum acceleration as σ_m is recommended by [33]. While the full derivation can be found in [33], for the limiting case, which assumes that the sampling time is much greater then the maneuver time constant $(\Delta \gg \tau_m)$, the covariance of the process noise can be written as in [34]

$$\mathbf{Q}_k = q \times \begin{bmatrix} \frac{\Delta^3}{3} & 0 & 0 & \frac{\Delta^2}{2} & 0 & 0 \\ 0 & \frac{\Delta^3}{3} & 0 & 0 & \frac{\Delta^2}{2} & 0 \\ 0 & 0 & \frac{\Delta^3}{3} & 0 & 0 & \frac{\Delta^2}{2} \\ \frac{\Delta^2}{2} & 0 & 0 & \Delta & 0 & 0 \\ 0 & \frac{\Delta^2}{2} & 0 & 0 & \Delta & 0 \\ 0 & 0 & \frac{\Delta^2}{2} & 0 & 0 & \Delta \end{bmatrix} \tag{3.1.25}$$

where $q = 0.001$ is an empirically derived process noise coefficient, which is set to a constant value to achieve a balanced velocity and position estimation.

4. Fusion Algorithm

Because the simulation runs using an invariant time interval for all sensors, and at any given time all sensors produce a measurement, which is assumed to come from the same track, the Kalman filter is initialized by *static values* in which the position comes from the state fusion based on the trace of the covariance matrix given in [35]. The velocity is an arbitrary, relatively small number that is calculated for a given aimpoint. The fusion method is based on the trace of the covariance matrix which is expressed as:

$$\hat{\mathbf{x}}(k) = \sum_{i=1}^{N} a_i \mathbf{z}_i(k) \tag{3.1.26}$$

where N is the total number of radars, $\mathbf{z}(k)$ is the measurement from the i^{th} radar at time k, and a_i is a quotient calculated as

$$a_i = \frac{1/tr(R_i(k))}{\sum_{j=1}^{N} \frac{1}{tr(R_j(k))}} \tag{3.1.27}$$

where $tr(\ )$ is the trace operator, and $R_i(k)$ is the mapped measurement covariance of the i^{th} radar at time k. The quotients a_i sum to unity by assumption.

The initial covariance, $P(0)$, is calculated using the measurement covariance matrix and multiplied by coefficient 10 to create a relatively large variance due to initializing with static values as

$$\mathbf{P}(0) = 10 \times \begin{bmatrix} \mathbf{R}_f & \varnothing \\ \varnothing & \mathbf{R}_f \end{bmatrix} \tag{3.1.28}$$

where $\mathbf{R}_f$ is the fused 3×3 mapped measurement covariance matrix, and $\varnothing$ is the 3×3 zero matrix. After initialization, the steps of the algorithm are as follows:

a. Prediction

$$\hat{\mathbf{x}}(k+1 \mid k) = \mathbf{F}(k)\hat{\mathbf{x}}(k \mid k) \tag{3.1.29}$$

$$\mathbf{P}(k+1 \mid k) = \mathbf{F}(k)\mathbf{P}(k \mid k)(\mathbf{F}(k))^T + \mathbf{Q}_k \tag{3.1.30}$$

b. Correction

$$\mathbf{K}_k = \mathbf{P}(k+1 \mid k)\mathbf{H}^T \left[\mathbf{H}\mathbf{P}(k+1 \mid k)\mathbf{H}^T + \mathbf{R}_{k+1} \right]^{-1} \tag{3.1.31}$$

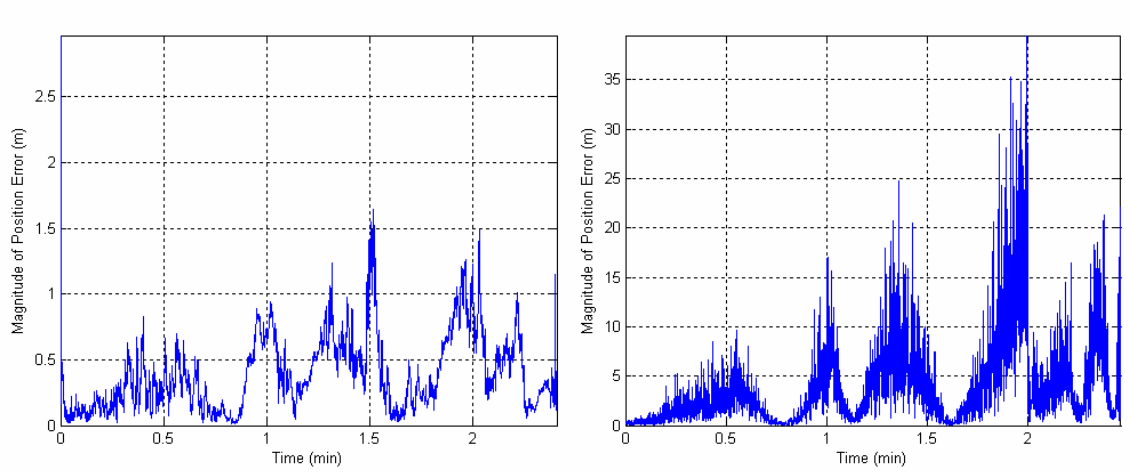
$$\hat{\mathbf{x}}(k+1 \mid k+1) = \hat{\mathbf{x}}(k+1 \mid k) + \mathbf{K}_k \left[\mathbf{z}(k+1) - \mathbf{H}\hat{\mathbf{x}}(k+1 \mid k) \right] \tag{3.1.32}$$

$$\mathbf{P}(k+1 \mid k+1) = (\mathbf{I} - \mathbf{K}_k\mathbf{H})\mathbf{P}(k+1 \mid k)(\mathbf{I} - \mathbf{K}_k\mathbf{H})^T + \mathbf{K}_k\mathbf{R}_{k+1}\mathbf{K}_{k+1}^T \tag{3.1.33}$$

where $\mathbf{K}_k$ is the Kalman gain at given time, and $\mathbf{I}$ is a 6×6 identity matrix.

5. Effects over Electronic Attacks

The main advantage of the Kalman filtering-based fusion algorithm is the reduction of the tracking position error. For the normal (without EA) case, the position error comparison is shown in Figure III-2. Because the initialization is from the static values, the initial position error is more than the average. The Kalman filter is able to converge quickly, thus minimizing the position error. The average miss distance is 10 m.



(a) (b)

Figure III-2 Position error introduced by the fusion center: (a) Kalman Filter, and (b) weighted average.

a. Effects against Jamming

The effect of the individual EA techniques while using the Kalman filter-based fusion algorithm is investigated next. Figure III-3 shows the tracking position error while being jammed. The resulting average miss distance is 12 m, which is 2 m higher than the nonjamming case.

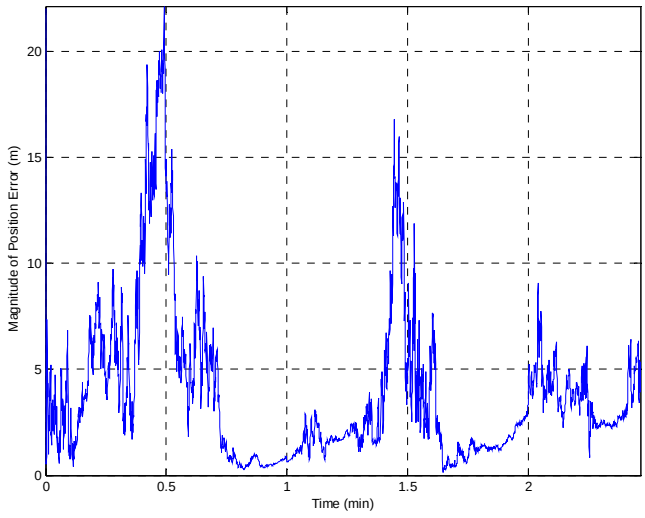


Figure III-3 Position error introduced by P_j = 10 kW and B_j= 4 GHz jammer

Figure III-4 and Figure III-5 show the effect of the Kalman filter, while the radar uses angle-only measurements for detection of the target. Using triangulation, the position error increased slightly, but it is still well below the non-Kalman case. The triangulation itself is a nonlinear transformation, so the covariance must be mapped again to compensate this side effect. The resulting average miss distance is about 15 m, and the convergence of the filter takes more time than unjammed cases.

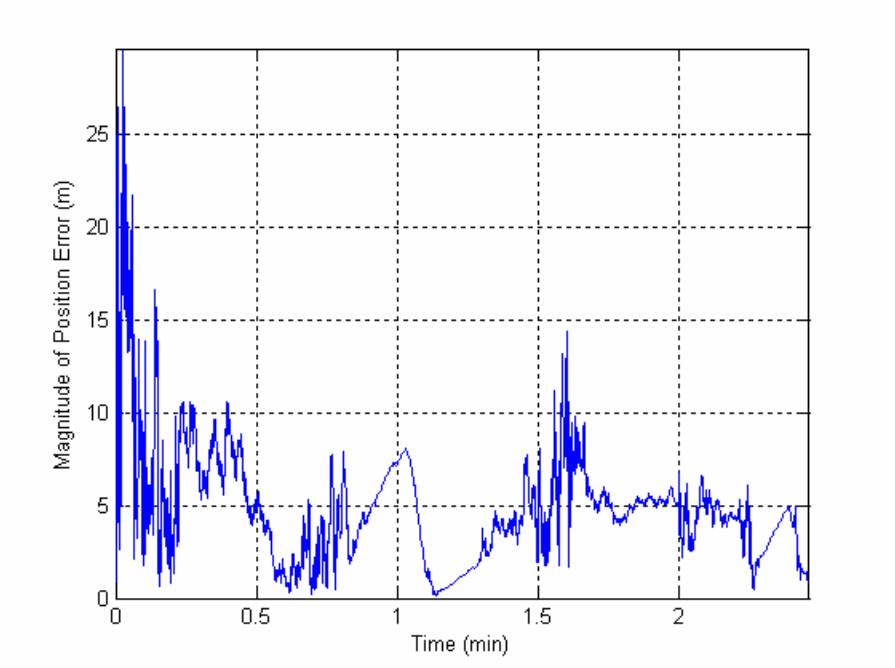


Figure III-4 The position error introduced by triangulation, while using Kalman filter based fusion algorithm.

As shown in Figure III-5, the lateral commanded acceleration is almost smooth and the achieved lateral acceleration follows it with a set time gap that is due to a third-order order system transfer function [5].

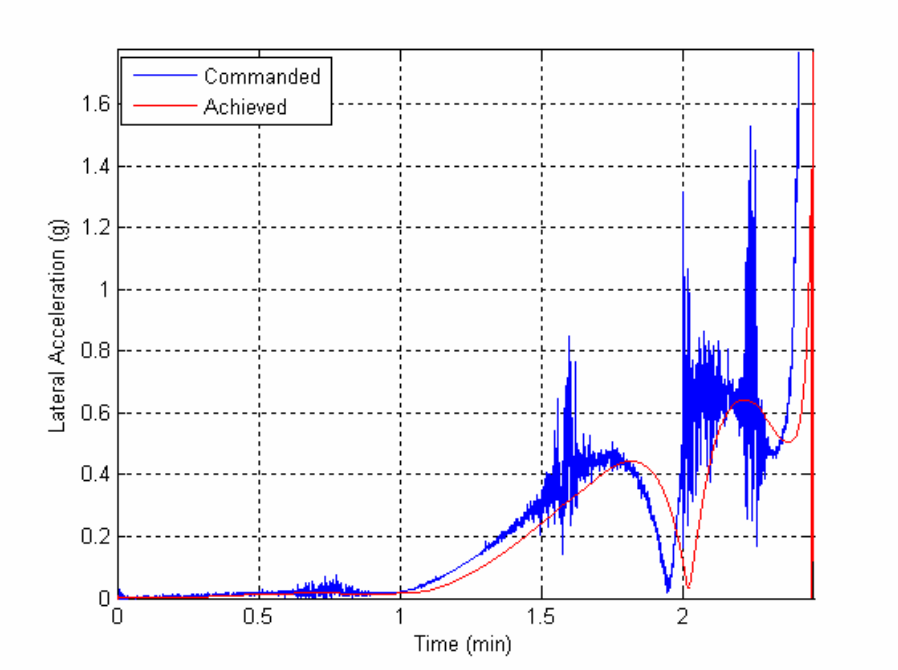


Figure III-5 The command lateral acceleration during the flight while using triangulation and applying the Kalman filter-based fusion algorithm.

b. Effect against RCS Reduction

The RCS reduction reduces the SNR of the target. A low RCS tends to increase the covariance of the RF sensor and the sensed tracking position errors. The uncertainties due to a low RCS cause the Kalman filter to be unstable and increase the error in the estimated position. The miss distance, however, does not change significantly.

Figure III-6 shows the position error when using the reduced RCS, Figure III-7 shows the command lateral acceleration during the flight. The increasing error, shown in Figure III-6, around 20 s and 90 s, is caused by the relatively low RCS, which is shown in Figure III-8.

The average miss distance is 15 m. Note that the position of the RF sensors is important here. The increased position error causes an increase in miss distance.

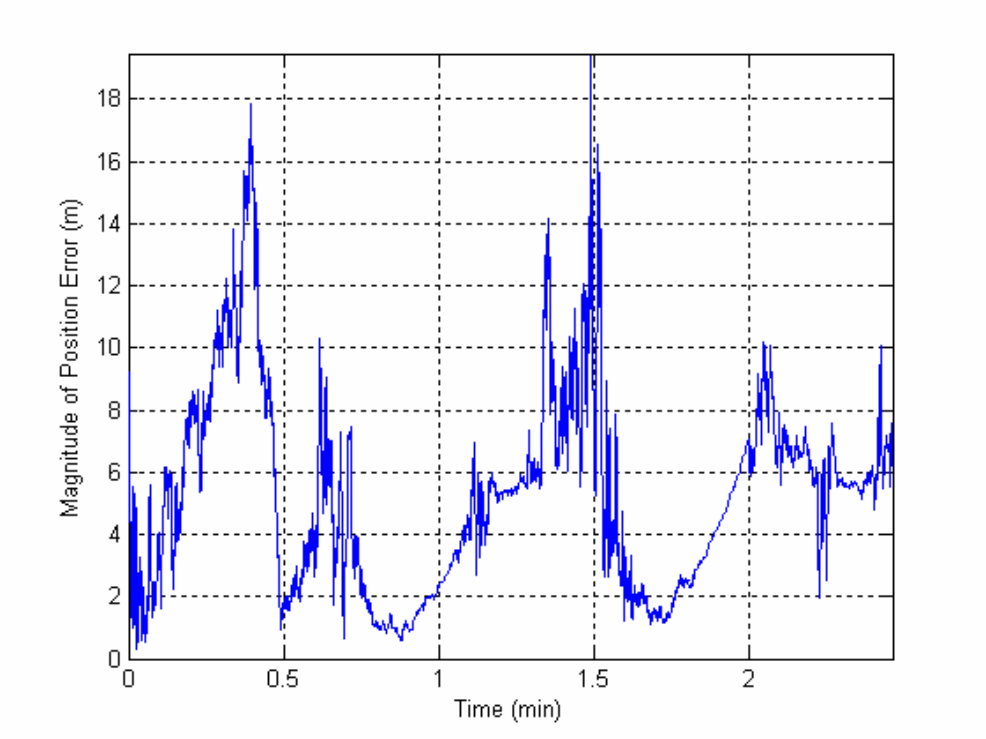


Figure III-6 Position error introduced by the RF sensor when the RCS is reduced.

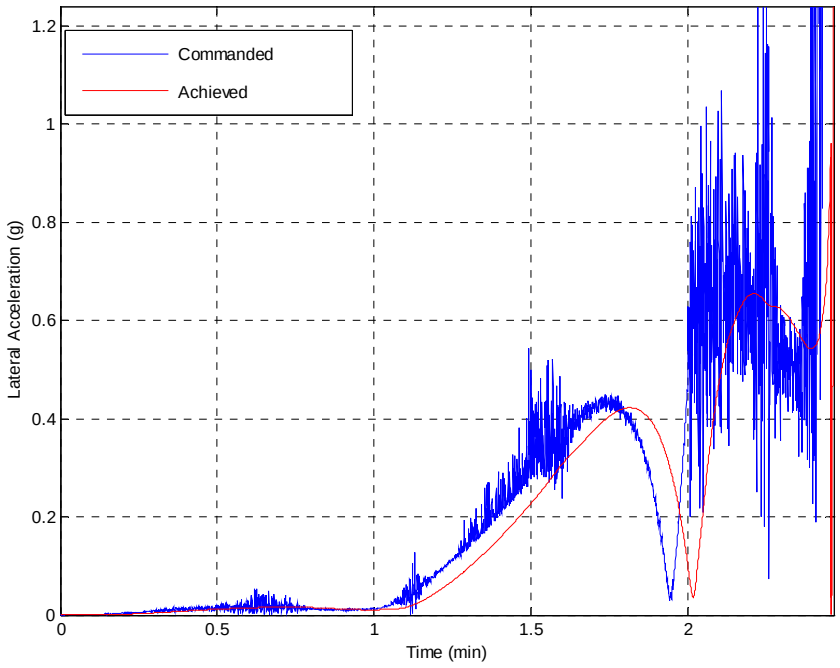


Figure III-7 Guidance command during the flight when using reduced RCS.

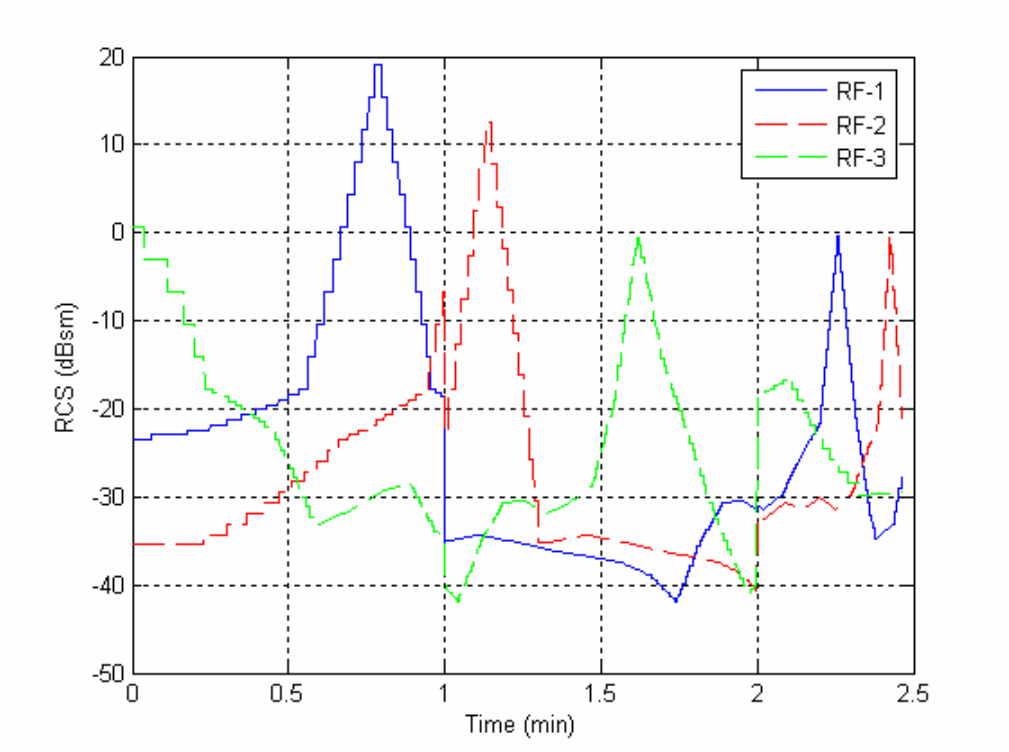


Figure III-8 RCS seen by the RF sensor of the simulation.

c. Effect against Chaff

The effect against chaff is less complicated. If the chaff-corridor entrance time is approximately the last phase of the interception, the miss distance caused by using the chaff corridor is slightly increased. Before that time, there is no observed increase caused by the chaff corridor.

d. Effect against Decoys

The effect of the decoys comes from the track transfer to the decoy and is also due to the robustness of the monopulse radar. The transfer time cannot be more than 12 s. The Kalman filter-based fusion algorithm is able to detect the deviation and correct it. Because the sensors get the information from the decoys, the covariance of the measurement will be different. In this case, the filter itself increases the gain and compensates the effect after a delay. Figure III-9 shows the tracking position error introduced when the decoys are released at t = 90 s. This is the perfect time for decoy release, according to [5]. The track reacquisition time is 20 s, which is way beyond our scenerio, but is used to test the algorithm. The resulting miss distance does not change significantly.

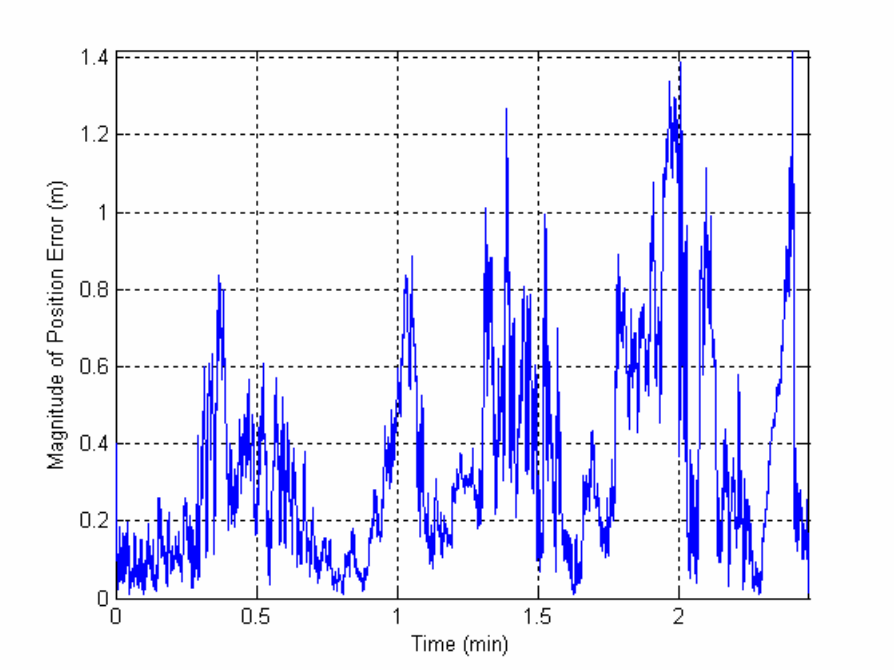


Figure III-9 Position error with decoy release.

e. Effect against Combined Attack

A combined attack might be applied by the target. It can include the use of jamming during the whole flight time, with a jammer that uses P_j = 10 kW and B_j = 4 GHz. The chaff corridor is created for t = 85 s over of 11 km, and the decoys are released just after the corridor exit time, with a reacquisition time of 20 s. The target is assumed to use a composite material that reduces the RCS. Because of the reduced RCS and the relatively high jamming power, the RF sensor is forced to use an angle-only measurement to create the measured position. This might be the worst-case scenario that a target missile produces. In reality, this scenario is unlikely to be applied because the power needed for the jammer is too large, and the required chaff rocket and decoys and the weight of the RCS reduction material, will degrade the missile performance

Figure III-10 shows the tracking position error caused by the combined attack, Figure III-11 shows the command lateral acceleration during the flight. The interception takes place 8 s later; the resulting average miss distance is 3,444 m. Even if this is an unlikely, worst-case scenario and the interceptor stays on course, the Kalman filter based fusion algorithm cannot mitigate the effect completely.

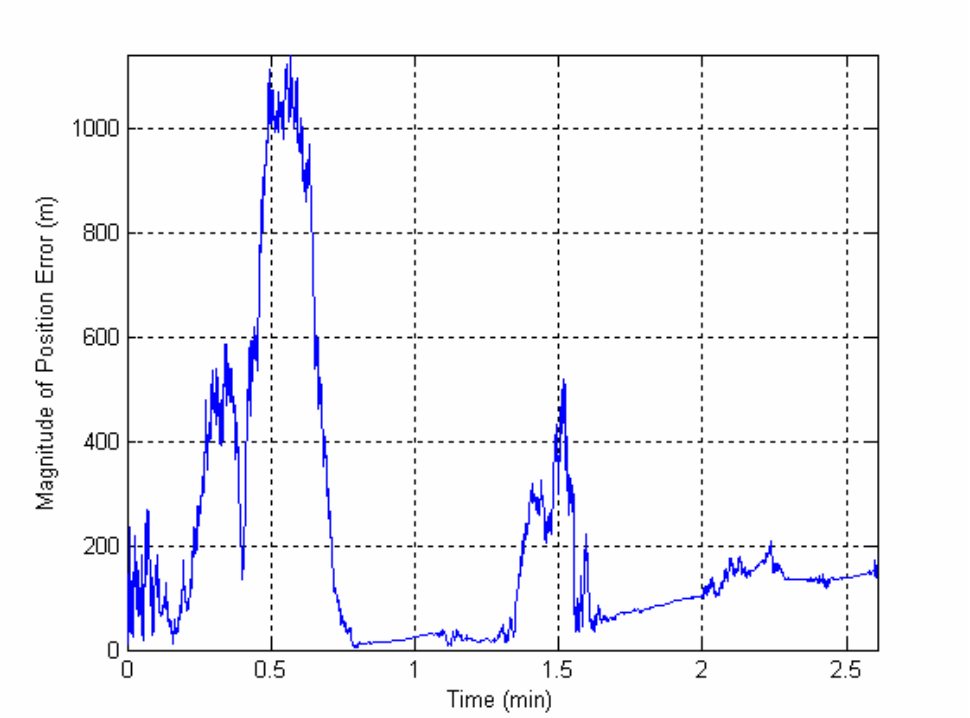


Figure III-10 Position error during combination attack.

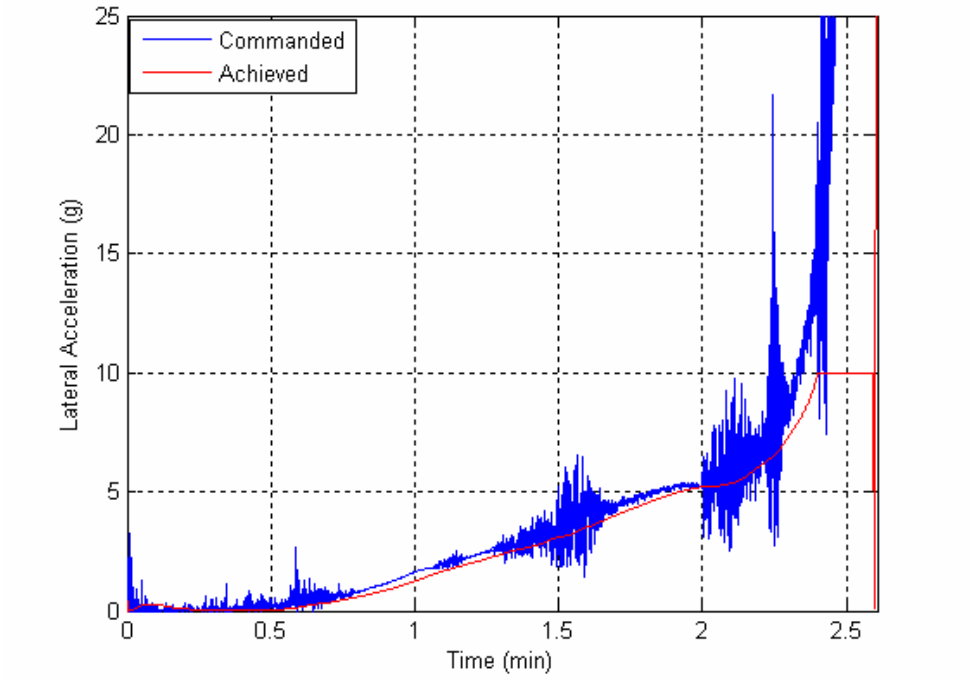


Figure III-11 The command lateral acceleration during the combination command.

B. BAYESIAN FUSION

In this section, we will investigate the Bayesian approach for the fusion system. A previous thesis [6] investigated the fusion algorithm for the boost-phase defense system and concluded that the Bayesian fusion is the best among all that were compared.

Although [6] explored the algorithm, data used were from the RF and IR sensors for input to the algorithm. Only RF sensors are investigated in this work, and the algorithm is revised to include three RF sensors. The sensor measurements are assumed to be Gaussian [6].

1. The Theory of Bayesian Fusion

The Bayesian approach is simply a *maximum a posteriori* technique to estimate the state of the target, given the measurement is ready for the time step k+1. This fusion technique requires the pdf for the unknown parameter $\hat{\mathbf{x}}(k)$, which is the target state estimation. The method can be formulated as

$$p(\hat{\mathbf{x}}_{k+1} \mid \mathbf{z}(k)) = \frac{p(\mathbf{z}(k) \mid \hat{\mathbf{x}}_k)\, p(\hat{\mathbf{x}}_k)}{\prod_i p(\mathbf{z}_i(k))} \tag{3.2.1}$$

where $p(\hat{\mathbf{x}}_k)$ is a priori pdf of the state estimation and $p(\mathbf{z}(k) \mid \hat{\mathbf{x}}_k)$ is the conditional posterior pdf of measurement given previous estimation. $\prod_i p(\mathbf{z}_i(k))$ is a normalizing product that is found from the measurement of the RF sensor, *i* indicates the i^{th} sensor, and *k* is the time step. The goal of the estimator is to seek a state estimation that maximizes (3.2.1).

The Bayesian fusion can be helpful in determining the probability estimate, given the measurement from the radar. It is implied that the Bayesian method has the ability to use subjective probabilities for the a priori probabilities, which leads us to conclude that the estimation is only as good as the input data [36].

The drawbacks of this method are [36]:

- Difficulty in defining a priori pdf.
- Complexity when there are multiple potential hypotheses and multiple conditional dependent events.
- Requirement that competing hypotheses be mutually exclusive.

- Lack of the ability to assign general uncertainties.

Finally, the computation complexity of the algorithm calculation is the main disadvantage of this approach.

2. Bayesian Fusion Algorithm

The algorithm uses three RF sensors that are not collocated. The pdf of the measurement is defined in [5] and [6] as Gaussian

$$p(\mathbf{z}_i(k)) \sim N(\overline{\mathbf{z}}, \mathbf{R}_k) \tag{3.2.2}$$

where $\overline{\mathbf{z}}$ is the first moment of the measurement vector, $\mathbf{z}_i(k)$, and $\mathbf{R}_k$ is the covariance matrix.

The conditional pdf of (3.2.1) posteriori can be written as

$$p(\mathbf{z}(k) \mid \hat{\mathbf{x}}_k) = \left|2\pi\mathbf{P}\right|^{-1/2} e^{-\frac{1}{2}(\overline{\mathbf{z}}-\hat{\mathbf{x}})^T \mathbf{P}^{-1}(\overline{\mathbf{z}}-\hat{\mathbf{x}})} \tag{3.2.3}$$

where the covariance comes from the state estimates.

The probabilities between the two points can be found from integration of (3.2.3). The illustration of the intersection of the three RF sensors pdf is shown in Figure III-12.

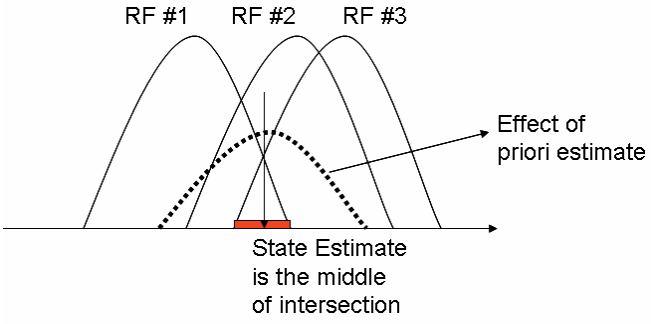


Figure III-12 The illustration of the pdf intersection volume

To achieve merging the previously used fusion algorithm, we use the same algorithm as in [6], find the random samples that are in the volume of the most probable intersection area, and calculate the estimation that gives the highest value for (3.2.1).

The position error introduced by Bayesian fusion is shown in Figure III-13. The tracking position error is greater than with the Kalman filter-based fusion algorithm for

normal tracking and the individual EA case, although the tracking ability is more stable. In Figure III-13, the normal case position error is shown for comparison.

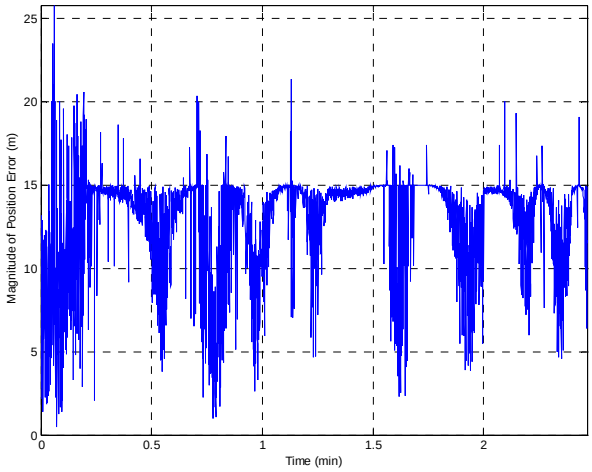


Figure III-13 The position error produced by the Bayesian fusion algorithm

3. Effect of Bayesian Fusion

The Bayesian fusion produces more stable results than the Kalman filter approach because it is less dependent on the measurement covariance than the Kalman filter. The Kalman filter is the best estimator that can be applied recursively; but because it is related to the covariance of the measurement and the plant noise, it can produce large errors when the distribution is non-Gaussian. The Kalman filter cannot converge in such cases and becomes unstable. The stability of the Bayesian makes it a good backup when the introduced track error is large.

In the worst case of using a combination attack by the target missile, the effect of the Bayesian is predictable. It reduces the position error within its stability range while for the other cases, the use of the Kalman filter may be considered.

The computation complexity of the Bayesian fusion algorithm makes it impractical for a real-time case. Producing the random sample matrices and choosing the best within them can be cumbersome if the measurement standard deviation is increased. The algorithm itself takes 27 s for a single time step in MATLAB®. If a total of 3,250 time steps are considered during full interception, the computation complexity can be easily

shown. The time consumption can be reduced if we use the sub-optimal Bayesian method. This method will be introduced in the next chapter investigating multi-target tracking.

Figure III-14 is the position error introduced by the Bayesian fusion algorithm when the target uses a combination of attacks described above.

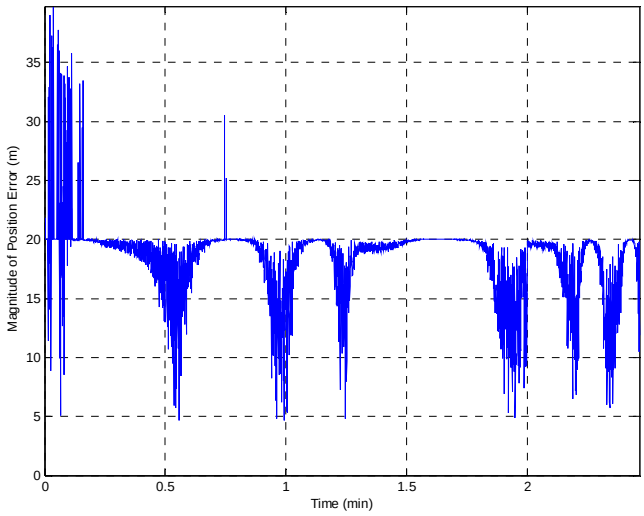


Figure III-14 The affect of the combination attack over Bayesian fusion.

Figure III-15 shows the command lateral acceleration during the flight. The resulting miss distance is 510 m, which well below the Kalman filter case.

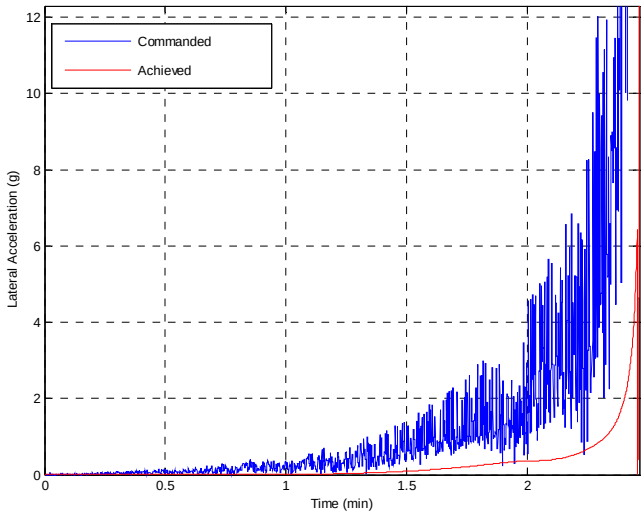


Figure III-15 The command lateral acceleration during the combination attack while using the Bayesian algorithm.

C. SUMMARY

In this chapter, an advanced fusion algorithm is introduced. The Kalman filter, which is the best estimator, using the least squares estimation, greatly increases the tracking capability. Individual EA effects are fully mitigated by this time-friendly algorithm. During a worst-case scenario, when the target uses a combination of attacks, the Kalman-based fusion algorithm fails.

Another advanced fusion algorithm is the Bayesian fusion, which is taken from [6]. The algorithm is more stable than the Kalman filter but too slow for use in real-time applications. A sub-optimal approach is needed to reduce the time needed for Bayesian algorithm. For this reason, we continue with the Kalman filter-based fusion algorithm to track the missile.

THIS PAGE INTENTIONALLY LEFT BLANK

IV. MULTI-TARGET TRACKING AND KILL VEHICLE REQUIREMENTS

In this chapter, we will address multi-target and KV requirements for the boost-phase intercept defense system. A single target against a single sensor is the simplest scenario for a tracking simulation, but this simple scenerio is not realistic. Multiple-target tracking (MTT) is an important requirement for a boost-phase defense system.

A. MTT SYSTEM

The basic MTT system block diagram is shown in Figure IV-1. Because the Kalman filter is optimal only when the observation used to update the track file is from the same target being tracked, it is important that track-measurement pairs be found correctly.

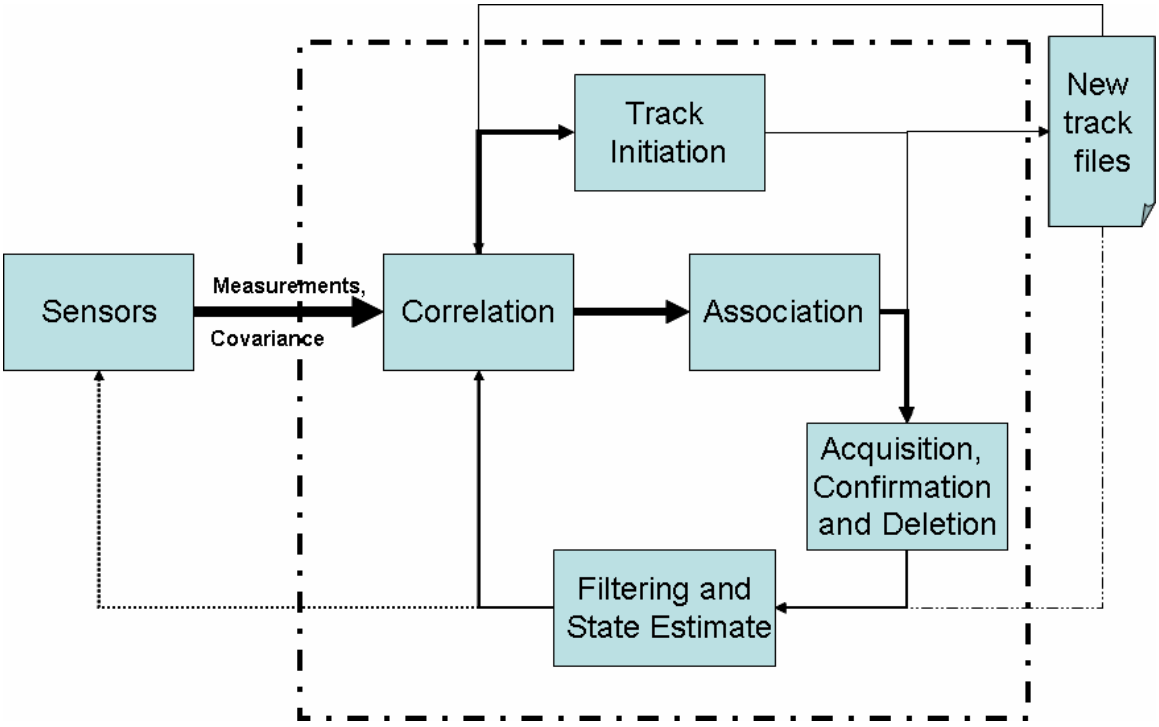


Figure IV-1 The block diagram of multi-target-tracking (MTT).

In the beginning, sensors are needed to detect the signal from the targets, and target discrimination must be performed to distinguish a ballistic missile target from all others. Then, the location of the target must be estimated, so that the multiple-target algo-

rithm can commence. The track initialization, correlation and association, state estimation and filtering, and track deletion can only be started after the signal is detected.

1. Target Discrimination

Target discrimination is not actually a part of an MTT algorithm. But since the objective of the boost-phase ballistic missile defense system is to intercept an ICBM launched toward friendly territory, we need to address discrimination. To achieve the objective, the system needs to differentiate a lethal echo from a nonlethal echo, and a friend from a rival, in the surveillance region. The modified schematic of the MTT block diagram should be as in Figure IV-2.

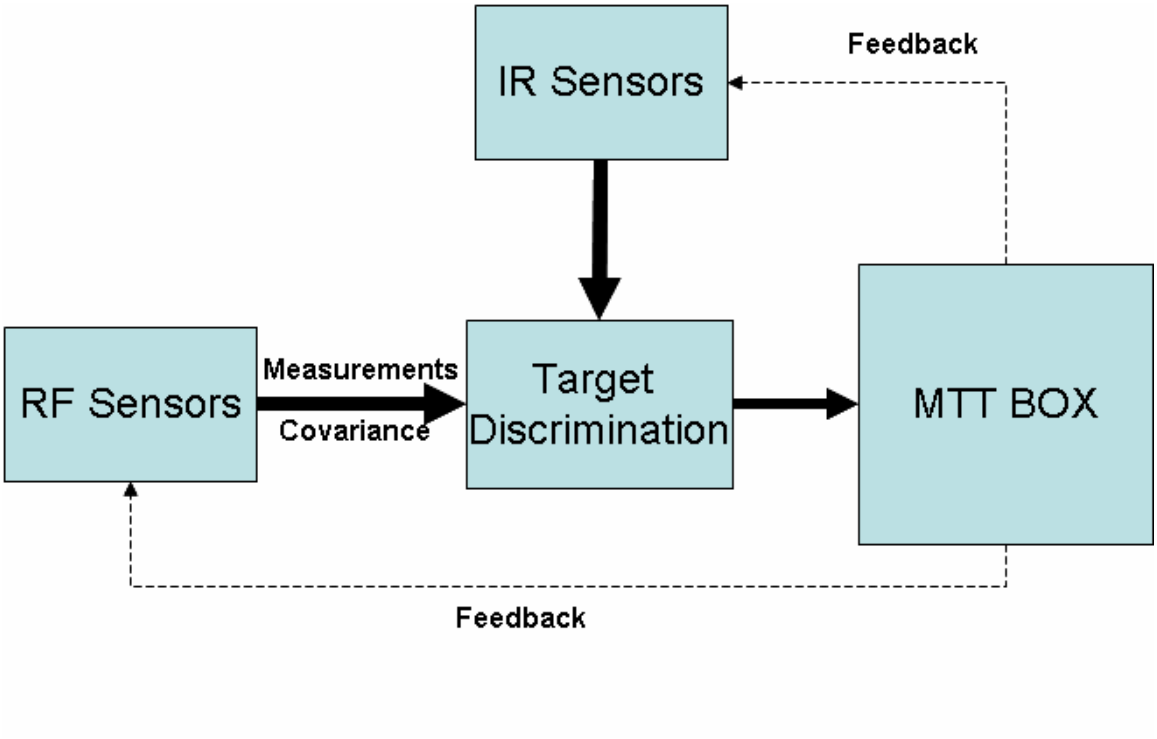


Figure IV-2 The suggested MTT block diagram with target discrimination.

Work concerning target discrimination has generally focused on the mid-course defense system. The discrimination algorithm differentiates a warhead from decoys or balloons in the exoatmospheric region. This is investigated in [37], [38], [39], [4], and [2].

References [37], [38], [40], and [2] discuss the IR sensors to discriminate the target in question while references [39], [38], [40], and [2] deal with the RF sensor to dis-

criminate the target. Reference [42] suggests using a combined "signal, feature, and decision level fusion" to discriminate, so that the hypothesized system is robust over noise and sensor degradation.

To discriminate the ICBM target at boost-phase, we need to analyze the features and characteristics of the target during the boost-phase in which we are interested. Then, we can differentiate the target among other echos in the area under surveillance.

Contrary to a mid-course target, we have an advantage with a boost-phase target, due to its high IR signature. The high temperature of the missile plume is easy to detect in the atmosphere in the related spectral band. In [5] and [6], the IR signature of the boost-phase target is investigated, and they concluded that a medium wave IR detector (3-5 μm) is needed to track the object, due to atmospheric transmittance. The ICBM plume has an approximate temperature of 1,035 K and has peak values that occur at a wavelength of ~2.8 μm. Figure IV-3 shows the spectral intensity of the Titan IIB missile, which has an approximate plume area of 600 m^2 [6].

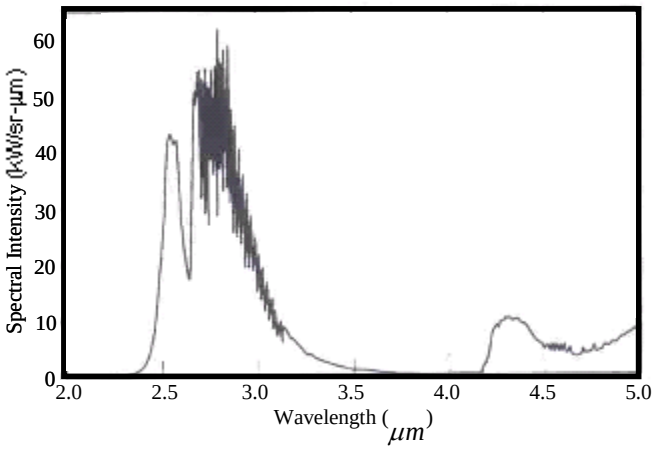


Figure IV-3 Spectral intensity of Titan IIB at an angle of attack of 7.4 deg (From [6]).

Because the peak radiation emitted at different temperatures can differ in wavelength and the missile body can also have a different IR signature, "using different spectral bands will be useful not just for observing any one object at multiple wavelengths, but also for determining an object's temperature" [4]. Table IV-1 shows the IR band for some important target plume temparatures.

Table IV-1 Peak radiation emitted by objects at different temperatures (From [4])

Temperature of Object (*K*)	$\lambda_{\max}(\mu m)$	**Radiation Band Used to Track Object**
1000	2.9	Short-wave infrared
675	4.3	Medium-wave infrared
450	6.4	Medium-to-long-wave infrared
300	10	Long-wave infrared
180	16	Long-wave infrared

Since we need to know the background effect at any level in order to increase the signal-to-clutter ratio and effectiveness of the discrimination algorithm, [37] suggests storing information beforehand to evaluate. For this purpose, the midcourse Space Experiment (MSX) satellite was launched into a 900 km polar orbit in 1996, to be "a long-duration, observatory-style measurement platform that collects several terabytes of high-quality data on Earth, Earth limb, and celestial backgrounds, ICBM-style targets, and resident space objects" [37]. The collected "phenomenology data" will be used to cover the gaps in a discrimination database [37]. Storing data on all types of flying-objects will help create a look-up table to compare with when needed.

An exact pattern of ballistic missile plume luminosity during its powered flight is given in [23]. Although this pattern does not belong to an actual threat, we can observe similar rocket plume peak patterns during their boost-phase.

Using IR sensors is not enough for discrimination purposes, even if the IR signature is strong. RF sensor discrimination should also be used to insure the target's configuration; hence we use the "multi-level sensor fusion algorithm for improved target discrimination" [42].

The RF sensor can use the RCS data that was investigated in [5], and in Chapter II of this work. The RF sensor can also use an integrated multiple model (IMM) type algo

rithm to discriminate the target in question. Using a hidden state to eliminate an unlikely echo, with the help of a "neural network-based sequential Bayesian classifier," is suggested in [40].

In addition to those signature types, the ICBM has a continuously increasing speed in its trajectory toward the aimpoint. The trajectory and the speed can be used to easily identify the ICBM among the other echos. Even if an echo comes from a commercial satellite launch that might use the same type of propellant, giving almost the same IR signature and speed, the trajectory estimation can be used to discriminate. All data collected by the radar and the IR sensor will help eliminate uncertainties.

Finally, "high range resolution techniques for ballistic missile targets" in the midcourse are used in [39]. They can be separated by 1 km at most. The technique suggested here can be modified for a boost-phase target that is not separated but that has tremendous speed as was already addressed in the paper.

2. Track Initiation

A definition of "new" tracks is given in [43]: "combinations of multiple sensor reports, separated in time, which do not correlate with any existing track, and provide reasonable kinematic parameters."

A track initialization can be divided into two steps:

a. Initiation of a tentative track when the measurements from a sensor cannot be paired with an existing track. This type of initiation can be used for any observations that are not in the correlation gate of previously initiated tracks (to confirm that the new track is really new).

b. The confirmation of a new track is a process in which the newly initiated track file is the real targets because a tentative initiation can be created for clutter and or false targets.

If any of the tracks are not updated with suitable correlated measurements, they become degraded and, therefore, will eventually be deleted [33].

The initiation algorithm should be implemented separately from correlation and association. There are recursive methods (e.g., NN-type algorithm, track split-type algo-

rithm, and M/N process), the two-point extrapolation method, the sliding window method, and the batch processing algorithm for different implementations [44].

No track initiation algorithms are implemented and addressed in the simulation. The track initialization is assumed to be done with the help of the IR sensors prior to the RF sensor's tracking for simplification.

3. Correlation

The correlation process assumes that the existence of an initial conditional state and a covariance of tracks at the time that a new measurement or a set of measurements is known. The measurement's covariance, which is accepted as unbiased and normally distributed with zero mean around the real target position, is well known.

Correlation is done to relate the observation or measurement from the set of RF sensors to the existing track files, or to initiate a new track file [45]. By applying correlation, we eliminate improbable measurement-to-track pairings [46] and determine which possible pairs are reasonable [34].

To construct a realistic simulation, all uncorrelated observations or measurements need to be cleared for every scan. "Elimination can be enforced by screening (prior to the generation of hypotheses) or by pruning (after generation of hypotheses). Gating, clustering, and classification is a screening technique; n-scan approximation is a pruning technique" [46].

Gating is a window constructed around the predicted measurement for a track; it is a very effective technique that reduces false correlations. Only observations whose measurements lie within the gate volume are associated with the existing track.

We assume that the motion of the target is described as in (3.1.6), with the plant noise $\nu(k)$ whose covariance is $\mathbf{Q}_k$, and the measurement is described as in (3.1.7) with the measurement noise $\omega(k)$ whose covariance is $\mathbf{R}_k$. The innovation, $\tilde{\mathbf{z}}(k)$, which is a vector of differences between the measured values and the predicted values can be found as [43], [33]

$$\tilde{\mathbf{z}}(k) = \mathbf{z}(k) - \mathbf{H}_k \hat{\mathbf{x}}(k+1|k) \tag{4.1.1}$$

where $\mathbf{H}_k$ is the measurement matrix at time *k*, and $\hat{\mathbf{x}}(k+1|k)$ is the predicted state of the target. It can be found as

$$\hat{\mathbf{x}}(k+1|k) = \mathbf{F}_k \hat{\mathbf{x}}(k|k) \tag{4.1.2}$$

where $\mathbf{F}_k$ is the known transition matrix, and $\hat{\mathbf{x}}(k|k)$ is the previous state estimation. The innovation variance is derived from the predicted covariance matrix, which should contain the variances of the quantities that are in the measurement vector. The innovation covariance can be found as in [33]

$$\mathbf{S} = \mathbf{H}_k \mathbf{P}(k+1|k)\mathbf{H}_k^T + \mathbf{R}_k \tag{4.1.3}$$

where $\mathbf{P}(k+1|k)$ is the predicted covariance of the state, and is found as

$$\mathbf{P}(k+1|k) = \mathbf{F}_k \mathbf{P}(k|k)\mathbf{F}_k^T + \mathbf{Q}_k$$

where $\mathbf{P}(k|k)$ is the previous state covariance.

For gating to work properly, the measurement's individual component should not be biased. When we map the measurement to the Cartesian coordinate system, due to the nonlinear mapping function the individual component will be biased. Hence, we need to use the extended Kalman filter to circumvent this situation.

The size and the shape of the gate can be defined in various ways. The exact definition is in [45]

$$D = \tilde{\mathbf{z}}_k^T \mathbf{S}^{-1} \tilde{\mathbf{z}}_k \le G \tag{4.1.4}$$

where D is the norm of the residual and G is the maximum likelihood gate, which can be defined as "a gate such that an observation falling within that gate is more likely from the track in question than from an extraneous source" [33]. The adaptive formula for the gate is given by [33]

$$G = 2\ln\left(\frac{P_d}{(1-P_d)\beta(2\pi)^{\frac{M}{2}}\sqrt{|\mathbf{S}|}}\right) \tag{4.1.5}$$

where P_d is the probability of detection, β is the new source density, M is the measurement dimension, and $|\mathbf{S}|$ is the determinant of the innovation covariance matrix.

The gate defined in (4.1.4) as the most general gate, which is known as the "ellipsoidal gate" [33], [46], and has the target in its center. The exact shape of the ellipsoid depends on the variance in the measurement dimensions. Figure IV-4 is a visualization of the ellipsoidal gate in three dimensions. The center is assumed to be at (1,000, 90, and 120), and the error bins describing the axis length are (100, 50, and 50) in the Cartesian coordinate system. If all error axes are equal to each other, then this will be a sphere.

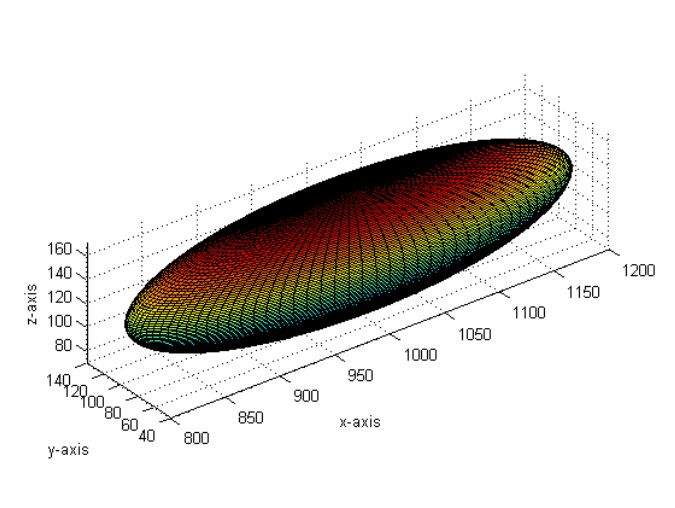


Figure IV-4 The general ellipsoidal gate in three dimensions.

The value of the gate thresholds can be found by using (4.1.5). Chi-square approximation is less computationally demanding, but it is also less adaptive.

If (4.1.4) can be transformed into its principal axes, which are off-diagonal, we reach the equation for the 3-dimensional case as:

$$q = \frac{(R_m - \hat{R})^2}{\sigma_R^2} + \frac{(\theta_m - \hat{\theta})^2}{\sigma_\theta^2} + \frac{(\phi_m - \hat{\phi})^2}{\sigma_\phi^2} \le G \qquad (4.1.6)$$

where R_m, θ_m, ϕ_m are the measured range, the elevation, and the azimuth with the variance of σ_R, σ_θ and σ_ϕ while $\hat{R}, \hat{\theta}, \hat{\phi}$ are the predicted parameter of the filter, respectively. All measurements are assumed to be normally distributed (i.e., $N(0, \sigma)$).

If each of the components is Gaussian distributed, then their squared sum (of three independent and identically distributed (IID) standard normal scalar-valued random variables) q is χ_3^2, which is a chi-squared random variable with three degrees of freedom. The probability, P_G, that a valid measurement will lie within the gate, G, can be found by the χ_3^2 table. For the case of three-dimensional measurements, the probability P_G can be expressed as [33]:

$$P_G = 2\frac{1}{\sqrt{2\pi}}\int_0^{\sqrt{G}} e^{-\frac{u^2}{2}}du - \sqrt{\frac{2G}{\pi}}e^{-\frac{G}{2}} \tag{4.1.7}$$

where the first part of the right-hand side of the equation is the standard Gaussian probability integral. The values of P_G for various cases are given in Table IV-2 [43] where M is the dimension of the measurement matrix, and G is the gate.

Table IV-2 Gate Thresholds and the probability mass P_G inside the gate (From [43]).

M \ G	**1**	**4**	**9**	**16**
1	0.683	0.954	0.997	0.99994
2	0.393	0.865	0.989	0.9997
3	0.199	0.739	0.971	0.9989

If we define the gate parameter separately for each dimension, such that

$$\begin{aligned}\frac{(R_m-\hat{R})^2}{\sigma_R^2} &\le G\\ \frac{(\theta_m-\hat{\theta})^2}{\sigma_\theta^2} &\le G\\ \frac{(\phi_m-\hat{\phi})^2}{\sigma_\phi^2} &\le G\end{aligned} \tag{4.1.8}$$

then we have defined the rectangular gate.

> The rectangular gate is the simplest gating technique that is easy to implement, because it tests the individual components of the measurements and needs less computations than ellipsoidal gating algorithm. But the el-

lipsoidal gating is more precise, because for a given probability P_G of the true measurement being in the gate, ellipsoidal gating requires a smaller volume than the rectangular gating [43].

Figure IV-5 is a representation of the final fact that, for a given gate size, the ellipsoidal gating will have a smaller probability of containing an extraneous observation.

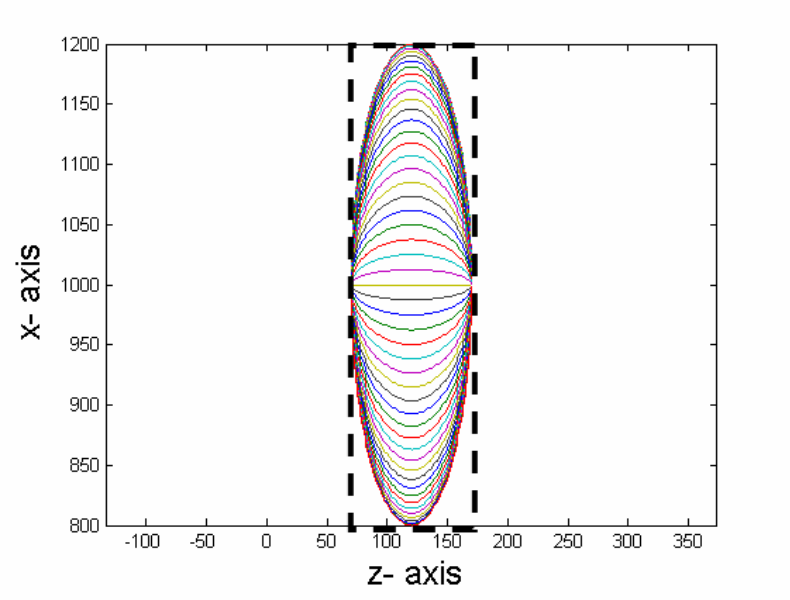


Figure IV-5 The rectangular versus ellipsoidal gating of previous visualization in two dimensions.

As a final point, while the volume of the rectangular gate is the pruduct of all the variances, the volume of a three-dimensional gate for the ellipsoidal case can be found from the formula [33]

$$V_G^3 = \frac{4}{3}\pi\sqrt{|\mathbf{S}|}G^{3/2} \tag{4.1.9}$$

4. Association

Data association is the second part of a MTT. Association is the process that solves the ambiguities arising from the correlation method. According to [33], the gating algorithm is only the first step of the MTT, and we need additional logic to determine an unambiguous measurement/track pairing.

Ambiguities arise when an observation lies within multiple track gates and/or when multiple measurements fall in a gate. Figure IV-6 is an illustration of both typical situations.

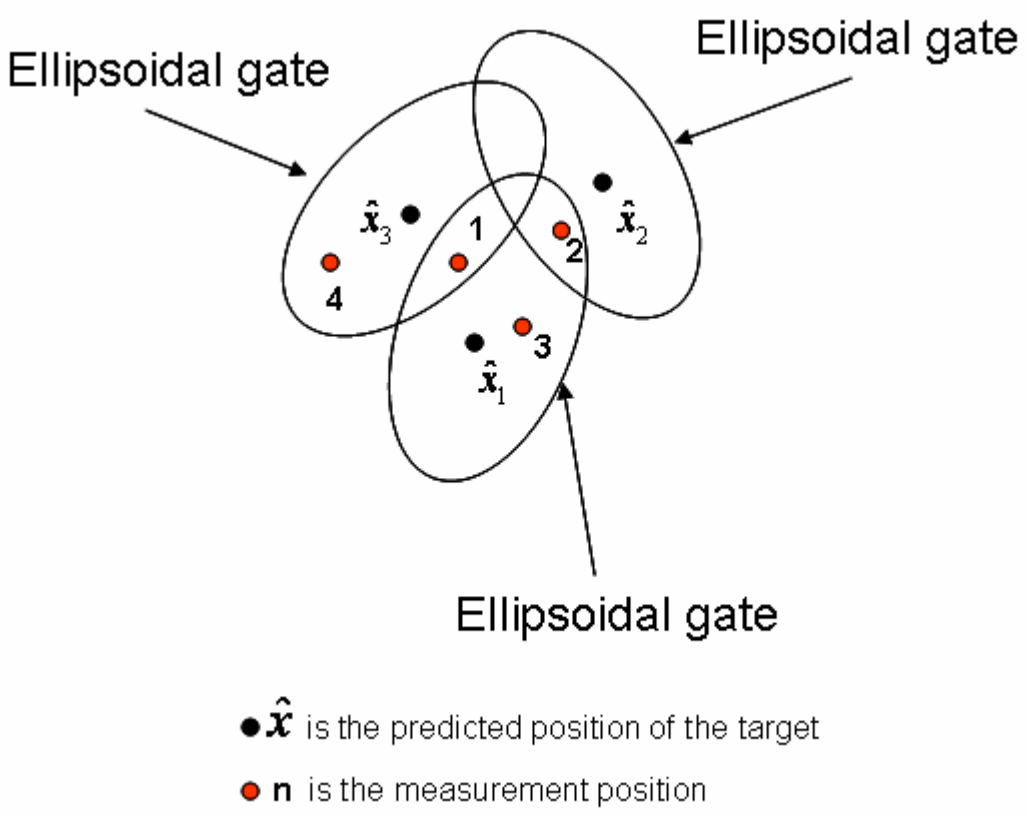


Figure IV-6 Ambiguities of the data association (After [33])

The association algorithm can be divided in to two categories: Bayes and non-Bayes [44]. The nearest-neighbor standard filter and the track-splitting algorithm, briefly described below, are non-Bayes-type approaches.

We can create an assignment matrix by using the example in Figure IV-6. We need to define a metric of measurement to evaluate the matrix. The likelihood function associated with the assignment of observation *j* to track *i* is given in [33] as:

$$g_{ij} = \frac{e^{-\frac{D}{2}}}{(2\pi)^{\frac{M}{2}}\sqrt{|\mathbf{S}_i|}} \qquad (4.1.10)$$

Maximizing the likelihood function means minimizing the statistical distance. We can find the statistical distances as in (4.1.4) using the weight of the innovation covariance. The assignment matrix helps us find the minimum $D = d^2(\mathbf{z})$ for every track. This method assumes that the probability of detection, P_d, is equal to unity. For a multiple-target case, the suggested method, nearest-neighbor standard filter (NNSF), will minimize the total statistical distance for all possible pairings [43], as is shown in Table IV-3.

Table IV-3 Assignment Matrix for Figure IV-6

	Measurements #1	**Measurements #2**	**Measurements #3**	**Measurements #4**
Track #1	9	X	4	X
Track #2	X	5	X	X
Track #3	8	X	X	3

For the optimal case, this algorithm gives the highest number of potential assignments, minimizes the total distance and finds the best global solution for the given time *k*. The combination (*m* choose *n*) that can be created from the matrix is found in [43]

$$\frac{m!}{(m-n)!} \tag{4.1.11}$$

where $m = \max(N_m, N_T)$, $n = \min(N_m, N_T)$, and N_m and N_T are the number of measurements and the number of tracks, respectively.

The total number of combinations can be increased exponentially when we track a large number of targets in a cluttered volume. In addition, P_d cannot always be equal to unity. For an increase in the number of combinations, we can also use a sub-optimal solution, as is suggested in [33] and [43]. None of the sub-optimal solutions, however, addresses the unity probability of detection.

The track-splitting algorithm is another way of association that uses every observation to evaluate the track for a given period of time. At the end of the period, the branch that has a maximum total likelihood or minimum total statistical distance is chosen. The other branches are deleted for a given track.

We chose to use the all-neighbors-type of approach, which uses a similar algorithm as the probabilistic data association filter (PDAF), to address the non-unity P_d. We assume that [43]:

- The number of targets is known or given beforehand.
- There is an existing track for every target or track initializing is done.
- A target can produce, at most, one measurement for a given sensor at a given time.
- A measurement could have come from, at most, one target, and any other measurements are false alarms/targets or clutter.
- False alarms/targets and clutter are independent and uniformly distributed.
- The conditional density of each target, given the past measurements, is Gaussian and independent across targets.
- Each target is widely separated spatially, such that correlation gate overlapping is unlikely.
- The elements of each (multi-dimensional) measurement are independent.

The PDAF algorithm is a sub-optimal Bayesian approach for association problems. The joint probabilistic data association filter (JPDAF) is extended for a multi-target case. The JPDAF/PDAF "is a relatively simple recursive method that does not require either the storage of past observation data or multiple hypotheses" [33] and is a special case of the multiple hypotheses tracking (MHT) algorithm. The MHT algorithm is described in [33], [43], [46], [47], [48], and [49].

In the case of a multiple-target scenario, the algorithm needs to address the cost of each possible measurement-track association, so we can find the feasible group, such that the cost of association is minimized [50]. The data association problem should be handled by computing the joint posterior probabilities of measurement association with multiple targets in clutter; and then, by using calculated probabilities, the sub-optimal Bayesian estimate of the target position is updated [51].

The false alarm distribution needed for the MTT algorithm is assumed to be a Poisson model with spatial density, λ, as

$$\mu_F(m) = e^{-\lambda\tilde{\mathbf{z}}} \frac{(\lambda\tilde{\mathbf{z}})^m}{m!} \tag{4.1.12}$$

where *m* is the expected number of measurements in the gate. For a nonparametric case, we can assume that [43]

$$\lambda = \frac{m}{V} \tag{4.1.13}$$

where *V* is the volume of the validation region given in (4.1.9).

The probabilities of the algorithm can then be found by accounting for a missing detection as

$$\beta_i(k) = \begin{cases} \dfrac{e_i}{b + \sum_j e_j}, & i = 1, \ldots . m_k \\ \dfrac{\mathrm{b}}{b + \sum_j e_j}, & i = 0 \end{cases} \tag{4.1.14}$$

where β_i is the probability that the i^{th} measurement is the correct measurement, m_k is the total number of correlated observations at time *k*, and

$$e_i = e^{-\frac{\tilde{\mathbf{z}}_i \mathbf{S}^{-1} \tilde{\mathbf{z}}_i}{2}}$$

$$b = \frac{\lambda\sqrt{|2\pi\mathbf{S}|}}{P_d}\left(1 - P_d P_G\right)$$

After computating the probabilities, the filter oparates as follows:

a. Combined innovation

$$\tilde{\mathbf{z}}(k) = \sum_{i=1}^{m_k} \beta_i \tilde{\mathbf{z}}_i \tag{4.1.15}$$

b. Filter gain

$$\mathbf{K}_k = \tilde{\mathbf{P}}\mathbf{H}^T\left[\mathbf{H}\tilde{\mathbf{P}}\mathbf{H}^T + \mathbf{R}_k\right]^{-1} \tag{4.1.16}$$

where $\tilde{\mathbf{P}}$ is the predicted covariance which is the same covariance used in (4.1.3).

c. State update

$$\hat{\mathbf{x}}(k+1|k+1) = \hat{\mathbf{x}}(k+1|k) + \mathbf{K}_k\tilde{\mathbf{z}}(k) \tag{4.1.17}$$

d. Covariance update

$$\mathbf{P}_k = \beta_0\tilde{\mathbf{P}} + (1-\beta_0)\mathbf{P}^c + \hat{\mathbf{P}} \tag{4.1.18}$$

where $\mathbf{P}^c$ is the track update covariance for a single-target case, as in (3.1.33), and $\hat{\mathbf{P}}$ is the effect of measurement origin uncertainty on the state covariance, accounting for the use of more than one original measurement defined as

$$\hat{\mathbf{P}} = \mathbf{K}_k\left[\sum_{i=1}^{m_k}\beta_i\tilde{\mathbf{z}}_i\tilde{\mathbf{z}}_i^T - \tilde{\mathbf{z}}\tilde{\mathbf{z}}^T\right]\mathbf{K}_k^T \tag{4.1.19}$$

5. MTT Algorithm Used

For simplicity, the following assumptions are considered:

a. There are only two targets at any particular time interval.

b. Target discrimination and track initializing are done before the tracking algorithm commences.

c. For tracking, there are enough RF sensors in the area.

d. The RF sensors can update the measurement every 50×10^{-3} s.

e. The RF sensors observe the target's range R, azimuth angle θ, and elevation angle ϕ with a Gaussian error if the targets are not jamming. When a jamming signal is present in the tracking space, the range information is suppressed, and the RF sensors switch to the angle-only measurement.

f. The SNR = 26.7 dB for a worst case. If the targets' RCS is reduced, the SNR = 10 dB.

g. The probability of miss, $P_m = 0.01$.

h. When the measurement originates from the real targets, the probability of mass inside the tracking gate, $P_G = 0.971$.

i. There are, at most, five false alarm/clutter hits that can be seen per target at the time of each observation.

j. False target/clutter is intentionally created from the source of the original radar measurement with a different standard deviation coefficient, ranging from 1 to 3, to force the PDAF algorithm. All other extraneous measurements are assumed to be gated out, using a rectangular gate with the same P_G .

The simulation uses the "*scope.m*" (see Appendix B) function for every radar to get the observation for the time *k*. The function uses the SNR value to determine the related standard deviation of the error. The observation output format for the normal case is given in Table IV-4

Table IV-4 The measurement output format of the "scope.m" function

Range
Azimuth angle
Elevation angle
RF position x-axis
RF position y-axis
RF position z-axis

The observation output format of the jamming case is given in Table IV-5. The range information is set to zero, because it is suppressed due to jamming.

Table IV-5 The measurement output format for the jamming case

0
Azimuth Angle
Elevation angle
RF position x-axis
RF position y-axis
RF position z-axis

The function measures the original target position, at most five false alarm/clutter measurements and decoys/chaff measurements. The latter is taken only when decoys/chaff are released from the target; then, all of them are stored in a measurement array. The RF sensor position in the Cartesian coordinate system is added at the end of each

measurement, so the fusion center can recognize which measurement comes from which sensor. In addition to the measurement array, the covariance of the measurement is produced in array format for use in the fusion center in the same manner.

Correlation, association, and fusion are implemented by the "*mash.m*" function (see Appendix B). The function uses a "*for-loop*" for every initialized target and applies ellipsoidal gating. The probabilities of the correlated measurements are computed, and the association is implemented by the equations given in the previous section.

We use spherical coordinate measurements to overcome the complexity of triangulation, when the RF sensors are jammed. This causes the use of a nonlinear function for the measurement, while the motion of the target is defined in the Cartesian coordinate system. To make the correlation, we need to find the innovation that is equal to the difference between the predicted position and the measured position of the target. To define the correlation, we need to define the extended measurement function to find the innovation. In a nonlinear case, the measurement is defined as

$$z(k) = \mathbf{h}(\mathbf{x}, k)\big|_{\mathbf{x}=\mathbf{x}(k+1|k)} \tag{4.1.20}$$

where

$$\mathbf{h}(\mathbf{x}, k) = \begin{bmatrix} \sqrt{x^2 + y^2 + z^2} \\ \tan^{-1}\left(\dfrac{y}{x}\right) \\ \tan^{-1}\left(\dfrac{z}{\sqrt{x^2 + y^2}}\right) \end{bmatrix},$$

$\mathbf{x}(k+1|k)$ is the predicted position of the target in question, and x, y, z are the Cartesian coordinates of the given position.

The measurement matrix $\mathbf{H}$ is the gradient of the measurement function, which is evaluated at predicted position as follows

$$\mathbf{H}_k = \begin{bmatrix} \frac{x}{\sqrt{x^2+y^2+z^2}} & \frac{y}{\sqrt{x^2+y^2+z^2}} & \frac{z}{\sqrt{x^2+y^2+z^2}} & 0 & 0 & 0 \\ -\frac{y}{x^2+y^2} & \frac{x}{x^2+y^2} & 0 & 0 & 0 & 0 \\ -\frac{zx}{\sqrt{x^2+y^2}(x^2+y^2+z^2)} & -\frac{zy}{\sqrt{x^2+y^2}(x^2+y^2+z^2)} & \frac{\sqrt{x^2+y^2}}{x^2+y^2+z^2} & 0 & 0 & 0 \end{bmatrix} \quad (4.1.21)$$

Then, the innovation will be

$$\tilde{\mathbf{z}}(k) = \mathbf{x}(k \mid k-1) - \mathbf{h}(x,k)\big|_{\mathbf{x}=\mathbf{x}(k|k-1)} . \quad (4.1.22)$$

All other steps for the correlation and the association will be the same as given in the previous sections above.

The estimated position and the related state covariance are used to get a fused final estimation. The fusion algorithm uses an inverse of the trace of the covariance matrix given in (3.1.26) and (3.1.27). The estimated position for every track with related state covariance is returned for future use.

The MTT algorithm uses ellipsoidal gating as correlation. The clutter, which can be excessively close to the target, can be located in the correlation gate. Because of this effect, an estimation of the state cannot be optimum. An estimated normal tracking position error is given in Figure IV-7.

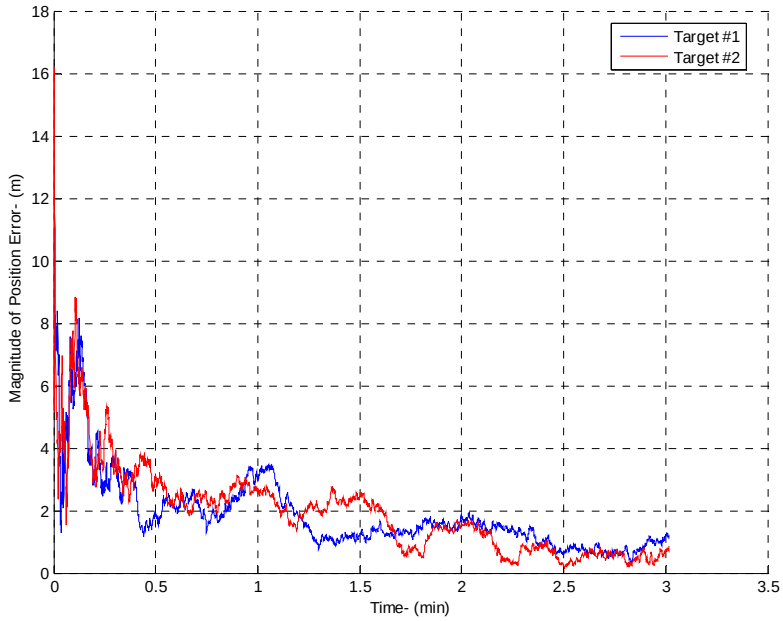


Figure IV-7 The MTT algorithm position error for a normal-case scenario. The plot is done for two targets.

When jamming is taken into account, the position error introduced by the fusion center due to angle-only measurement is shown in Figure IV-8. At the end, the error is compensated.

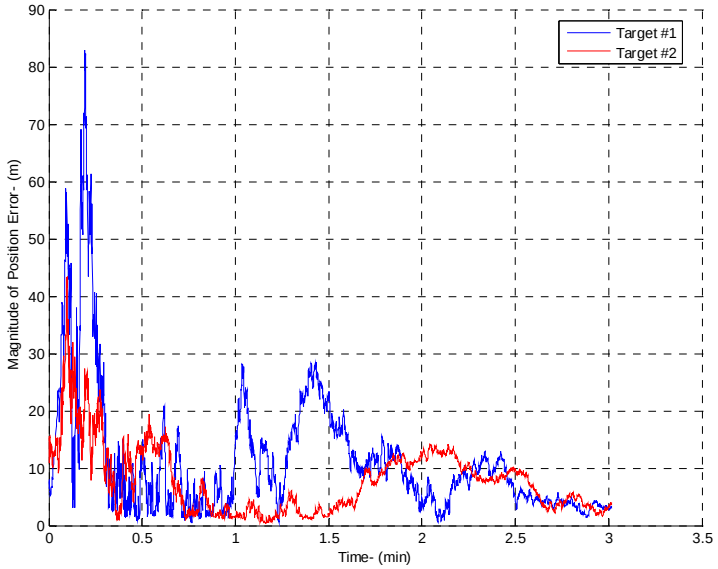


Figure IV-8 The MTT algorithm target position error when it is jammed by both of the targets: P_j = 10 kW and B_j= 4 GHz.

The reduced RCS effect on the fusion center is shown in Figure IV-9. The algorithm uses the reduced RCS and adjusts the gain of the filter to overcome it. At the end of the boost phase, the position error caused by the reduced RCS coating is reduced below 5 m, as in the jamming case.

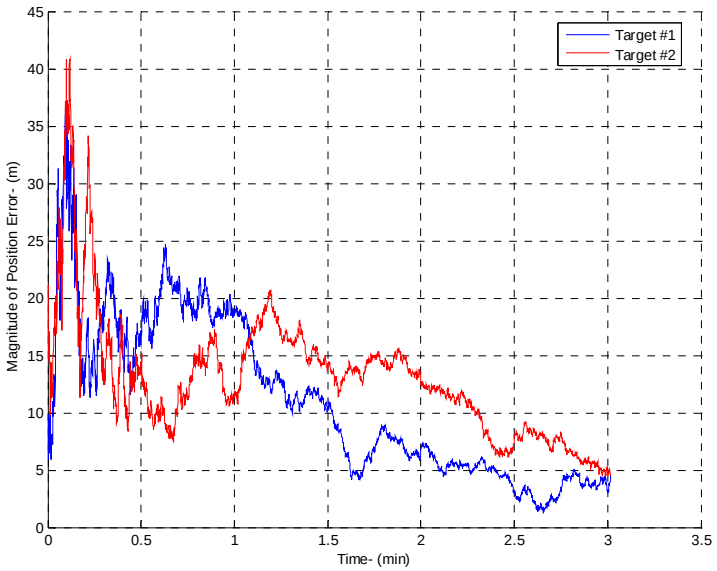


Figure IV-9 MTT algorithm target position error when both targets use reduced RCS coating.

Because the correlation algorithm already takes into account decoys and chaff when they are released, their effects are less than the Kalman filter-based fusion center algorithm shown in Chapter-III.

The combined-attack effect, described in Chapter-III, is shown in Figure IV-10. While the error is high at the beginning, the algorithm achieves a reduction in error by the end of the boost phase. The MTT algorithm, which uses the sub-optimal Bayesian technique called PDAF, is more stable and less error-prone than the Kalman filter, defined for a single-target case in Cartesian coordinates in Chapter III. A miss distance analysis will be conducted along with the guidance acceleration analysis in the next section while evaluating the kill vehicle.

Figure IV-10 MTT algorithm position error, when the combined attack is used.

B. KILL VEHICLE

In this section, we will discus the kill vehicle that is carried in a boost-phase ballistic missile interception. The purpose of this investigation is to give the basic requirements about KV. At the end, we will implement and analyze the effect of a KV's *end game* in our simulation.

All of the "KVs" recently reported are designed for mid-course intercept. Here, we examine the requirements of boost-phase kill vehicles.

1. Evolution of Kill Vehicles

The "Exoatmospheric Kill Vehicle (EKV) is a small flying device located in the tip of a Ground-Based Interceptor (GBI) missile. It is designed to separate from the GBI in flight and punch through the Earth's atmosphere, and smash into the incoming ballistic missile" [52].

Since the first rocket launched in World War II, all military strategists have intended to intercept an incoming missile on its way to the impact point [53]. During the

last thirty years, homing-style interceptors have had enough accuracy to succeed in colliding with an incoming missile [23]. The milestone of the progress can be summarized as follows [54].

a. September 8, 1944: The Missile Age begins with German V-2 missiles.

b. July 1945- March 1946: Military officers recommend that the U.S. start researching and developing ways to defend against incoming ballistic missiles.

c. 1957-1958: The U.S. steps up its missile defense efforts after the Soviets test Sputnik launch and begins work on the Nike-Zeus system, its first major anti-ballistic missile system.

d. July 19, 1962: One of the Army's Zeus missile interceptors, launched from Kwajalein Atoll in the Marshall Islands, comes within 2 km of a mock warhead that had been fired from California.

e. November 10, 1966: The U.S. publicly confirms that the Soviet Union is deploying its anti-ballistic missile (ABM) system around Moscow, which had been first detected two years earlier.

f. March 23-25, 1983: The U.S. President Ronald Reagan, in a nationally televised address, calls for research and development of missile defenses.

g. June 10, 1984: After several failed attempts, the Army successfully tests its Homing Overlay Experiment (HOE) in which a “kill vehicle” launched on top a booster rocket from Kwajalein Atoll in the Marshall Islands homes in on its target, an ICBM.

h. January 29, 1991: The Pentagon's Exoatmospheric Reentry Vehicle Interceptor System (ERIS), which was built using technology from the SDI's HOE, reportedly hits and kills a mock target in space.

i. June 24, 1997: The first “fly by” test of the National Missile Defense system is conducted.

j. October 2, 1999: The first hit-to-kill test of Clinton's NMD system is conducted. Despite initial problems with its telescopes, the kill vehicle is able to locate the warhead and collide with it.

k. March 15, 2002: Integrated Flight Test (IFT-8) is successfully conducted by interception of a ballistic missile target for the GMD program [3]

For the purpose of the kill vehicle, the HOE and ERIS projects are an important milestones. Both of them are examples of direct ‘hit-to-kill” interceptor.

The HOE is a

Vehicle consisted of the first two stages (Thiokol M55E1 + Aerojet General M56A1) of a LGM-30A/B Minuteman I ICBM, which boosted a large

KKV (Kinetic Kill Vehicle) to high altitude. The KKV was equipped with an IR seeker, guidance electronics and a propulsion system. Once in space, the KKV could extend a folded structure similar to an umbrella skeleton of 4 m (13 ft) diameter to enhance its effective cross section [53].

Figure IV-11 shows the defined KV for HOE.

Figure IV-11 The HOE (From [53]).

The second important step in U.S. missile defense is the test of the ERIS.

The ERIS test missiles consisted of the second and third stage (Aerojet General M56A1 + Hercules M57A1) of surplus LGM-30A/B Minuteman I ICBMs, which boosted the hit-to-kill interceptor vehicle into space. Sensor and guidance technology of the ERIS KKV was based on the experience won by the earlier HOE (Homing Overlay Experiment) tests. Because of technology improvements the ERIS KKV, which used an inflatable octagonal “kill enhancer”, was significantly smaller and lighter than the HOE KKV [56].

Figure IV-12 shows the defined KV for ERIS. The KV’s size and weight get smaller and smaller as the key technology of sensors, motion adapters and data-processing devices are developed [23].

Figure IV-12 The ERIS (From [56])

Finally, in 1997 and 1998, after the evaluation of the test results, “Raytheon was selected as the prime contractor for the development of the EKV for the operational GBI missile” [1]. The benefits of the newly designed EKV are summarized as follows [57]:

a. adds synergy to a multi-layered defense,

b. counters the threat in the midcourse phase of flight,

c. target selection made in presence of multiple decoys,

d. “hit-to-kill” technology allows complete destruction of weapons of mass destruction, and

e. Payload consists of EKV and adapter for booster.

Figure IV-13 shows the developed KV for the midcourse.

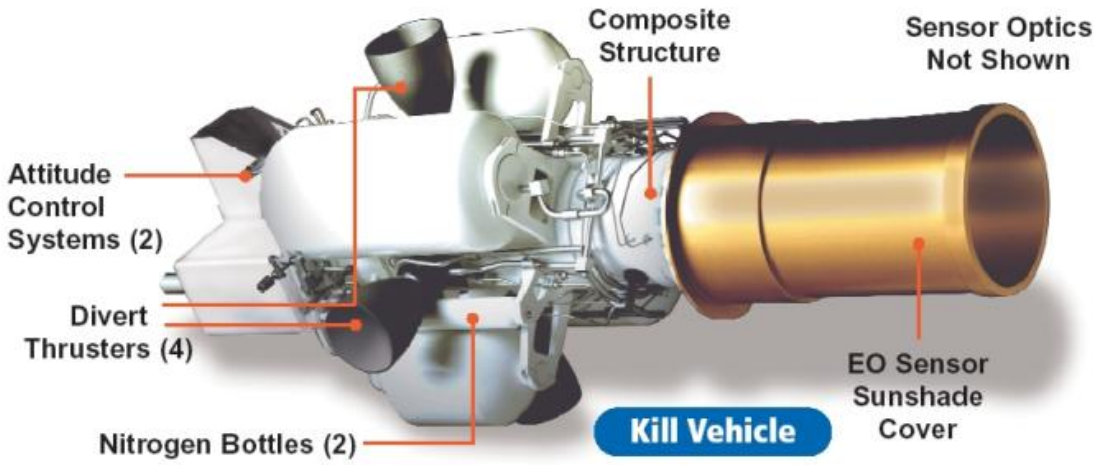


Figure IV-13 Raytheon EKV (From [57])

For the KV configuration, the *cruciform* is mostly chosen to minimize the size and the weight of the KV [23]. The four divert thrusters are mounted in the form of an "X" to give sensitivity in maneuvering. Owing to the center-located fuel tank, the center of gravity is not changed significantly when the thruster consumes the fuel.

Until now, all KVs have been for the midcourse interception. While most of the design factors are the same, there are some differences in boost-phase KVs. The main difference is in the weight and the sensor types. Although, the "wet weight" (i.e., fully loaded with fuel) of the EKV for boost-phase is around 150 kg [23], the "data sheet" [57] indicates the weight of the Raytheon EKV is approximately 140 pounds. The weight difference comes from a need for an excess divert capability of the boost-phase KV.

2. On-Board Sensors

The sensor on a KV can be compared to the "eye" of a human being. The sensors typically consist of one or more seekers that acquire the target and help guide the vehicle to the interception point [58].

The selection of sensors depends on the target in question. Unlike a midcourse target, in the boost-phase target, which is assumed to use solid propellant, a plume of exhaust gases creates a smoke screen, which affects the IR seekers, depending on their approach angles [58]. The high intensity of the exhaust gases helps detection, but for discrimination purposes the seeker must detect the missile's body temperature.

For the IR seekers, using the on-board SWIR seeker to track the plume is a solution suggested by [23]. For the IR seekers to work accordingly, the atmosphere must be thin enough, and the surface of the seeker's optical telescope needs to be cooled, as in the "Raytheon EKV." The choice of a multiple-waveband IR seeker for the Raytheon EKV might be adopted for the boost-phase KV, even if it is more complicated than the SWIR seeker.

The KV's IR seeker has advantages over space-based IR sensors with respect to the background clutter. Most of the time, an IR seeker looks up toward empty space, which has a very cool background temperature. The signal-to-clutter ratio can be more than adequate for tracking. According to [23], eventhough the seeker looks directly through the sun, the suggested IR seeker can detect the plume.

For the purpose of *hardbod*-plume discrimination and to help to in the choice of IR seekers for the KV, a Near Field IR Experiment (NFIRE) satellite is planned for launching in 2006. "Data from the Near Field IR Experiment will help validate the MDA's choice of KVs and tracking sensors for boost-phase missile defense, and help improve the guidance and homing ability of ground-based interceptors" [59].

The second sensor proposed the KVs is an active light detection and ranging (LIDAR) system [23] and [58], which is more accurate than the IR seekers. The LIDAR can measure the range to the target. With the help of passive imaging seekers, the LIDAR system can help improve fire control and weapon system applications to allow target acquisition, tracking, classification, and imaging [60].

The range equation of the LIDAR to detect the point mass target is given in [60] as

$$R = \sqrt[4]{\frac{E_T \sigma D^4 \eta_{ATM} \eta_{SYS}}{16(SNR)\lambda hc}} \quad \text{m} \tag{4.3.1}$$

where E_T is the transmitted power, D is the optical aperture diameter in m, η_{ATM} is the atmospheric transmission factor, η_{SYS} is system transmission factor, SNR is signal-to-noise ratio, and $h = 6.626 \times 10^{-34}$ J-S is the Planck's constant. While the range is directly proportional to the aperture size, there is a limitation on the maximum aperture size.

The point resolution standard deviation is given by [60] as

$$\sigma_\theta = \frac{3\pi}{16}\frac{\lambda}{D}\frac{1}{\sqrt{SNR}}\frac{1}{S_T} \quad \text{rad} \tag{4.3.2}$$

where S_T is a complex function related to the image's spatial distribution and antenna obscuration. For an unobscured aperture, this value can be accepted as unity.

The characteristics of the space laser are listed in Table IV-6. From there, [23] chooses the Fibertec device. Depending on the values given in [23], the accuracy of the LIDAR is assumed to be 0.5 m at the final stage of interception, which is much less than the diameter of the target of interest here.

Table IV-6 Characteristics of the Space Laser (From [23]).

Laser	**Efficiency**
Mars observer Laser Altimeter (1.06 μm, 40 mJ, 10 pps)	3%
Vegetation Canopy Laser (1.06 μm, 15 mJ, 10 – 240 pps)	6%
Fibertek proposal for improvement 1.06 μm laser	10%
Nd:YAG slab 1.06 μm, 808-nm diode pump, 100 W)	6%
Yb fiber (1.03-1.10 μm, 100 W)	6%-8%

To improve the laser pointing accuracy, a joint observer-based adaptive controller is suggested by [61]. With the help of a controller, system nonlinearities, such as KV dynamics, acceleration limiters, laser properties, and interaction between the KV and overall tracking system might be subjugated.

In contrast to LIDAR, microwave radar or the state-of-art *low probability of Intercept* (LPI), radar can produce the same accuracy. LIDAR can be affected by *alumina particle* in the produced plume, depending on the angle of attack, due to the use of solid propellant. While range resolution is the same as LIDAR (0.5 m) within the last few seconds, a 15-cm aperture radar with an aperture efficiency of 0.8, working at 10 GHz, using a pulsewidth of 16 μs, can produce a resolution cell of 0.54×0.54 m at 10 km by assuming, at least, the use of a 20-dB single pulse SNR .

If the problem with complexity, due to the increased numbers of FFTs required, is solved, FMCW LPI radar can also be used for a same size resolution cell, using only a fraction of the power [62]. Using the "LPI Tool Box" supplied in [62], the parameter of the LPI radar for a 0.5-m-resolution is shown in Table IV-7.

Table IV-7 FMCW LPI radar parameter for 0.5 m resolution at 10 GHz.

Doppler Frequency (Hz) f_d	612×10^{3}
Max delay (s) t_d	10×10^{-3}
Coherent processing interval (s) t_0	45×10^{-3}
Spectral width (Hz) $\Delta\omega$	222.2×10^{3}
Effective transmitted modulation bandwidth (Hz) $\Delta F'$	245.45×10^{6}
Degraded (effective) Resolution (m) $\Delta R'$	0.61
Maximum beat frequency (Hz) $f_b^{\max}$	55.15×10^{6}
Minimum sampling rate of ADC (Samples/s) f_s	110.3×10^{6}
FFT size N'	65,536
Adjusted sampling rate of ADC (Samples/s) $f_s^{'}$	145.63×10^{6}
Unambiguous max Beat frequency (Hz)	72.8×10^{6}
Time bandwidth product (Hz/s) $t_0\Delta F'$	110.45×10^{3}

3. Kill Vehicle Requirements for Intercept

The requirements for a successful interception depend on the phases of flight that are given below. The phase of the interception is divided into four sections by [23]:

a. Interceptor boost: the phase between the launch of the GBI and the end of all boosters. In this phase, accuracy of the sea- or space-based sensors to guide the interceptor to the correct position is the only requirement. Depending on the accuracy of the guidance, the KV can be fired in a certain direction so the initial divert of the KV becomes smaller. Because this

phase affects the later phases, the filter that estimates the target position is very important.

b. KV divert: the phase between the kill vehicle launch and the on-board sensors' acquisition of the target. If the interceptor boost phase is accurate enough, the KV can acquire the target easily. If the position of the target is much farther away than the expected position, the divert velocity and the time for acquisition will increase. The velocity change capacity of the KV is a key element in this phase. To make the final correction, the total velocity change requirement is found to be 2.5 km/s for a closing velocity V_c in the range of $10-14$ km/s [23].

c. KV homing: the phase between the acquisition of a target with an on-board sensor and the final stage of the interception. In this phase, a basic requirement of interception is the accuracy and maneuver capacity of the KV. Because the intended boost-phase target continues to accelerate, the navigation adapter has to cope with this acceleration along with any possible evasive maneuver of the target. To achieve this, [23] assumes that a maximum 15g capacity of the KV will be enough. This assumption is true only if the on-board sensor is accurate enough (i.e., within 0.5 m standard deviation, as was given in the section above).

d. Endgame: the final stage of interception. "The KV must have sufficiently precise and timely information on the target's current and probable future state and sufficient responsiveness and acceleration to hit the target" [23].

In addition to accurate position information and the maneuver capacity of the kill vehicle, weight consideration is also an important factor for successful interception. A suggestion about the weight consideration is made by [23]. Using an SWIR and a LIDAR as on-board sensors and a divert thruster for a total divert of 2.5 km/s, the estimated weight of the KV is given in Figure IV-14.

KV Segment or Subassy	Existing Technology KV (kg)	LLNL Space-Based ATKV (kg)	Surface-Based Baseline KV (kg)	Space-Based Baseline KV (kg)	Notes
Divert and Attitude Control System (DACS)					KV sized for closing velocities of 10-14 km/sec. Total time of KV operation 120 sec. Assumes 4 divert thrusters in cruciform configuration, sized to deliver 15 g in last 10 sec. LLNL example uses fixed thruster mass regardless of divert requirements; baseline KV corrects for this by adjusting divert thruster mass for 15 g using Wilkening scaling factors. ACS impulse assumed to be 5% of divert impulse.
Pressure regulator	Included	2.00	0.50	0.50	
Divert Thrusters	17.15	3.00	10.09	9.77	
ACS Trusters	1.50	included	1.01	0.98	
Valve drivers	included	included	included	included	
Manifold	included	included	included	included	
Subtotal, DACS	**18.65**	**5.00**	**11.60**	**11.24**	
Seeker (less IMU)	4.90	7.00	7.00	7.00	Includes LIDAR and passive IR and visible sensors. Used Clementine sensor suite masses
Contingency for shielding for focal plane array			1.00	1.00	See Natural and Induced Environment section
IMU (Inertial measurement unit)	1.50	1.00	1.00	1.00	
Avionics	11.50	8.00	8.00	8.00	Avionics include: guidance/control computer,tactical communications transponder, KV electronic safe arm, FTS antenna (non-tactical), FTS battery, command destruct receiver, signal conditioner/submux, X-band antennas, X-band transmit module, power divider/hybrid coupler, J-box, control module, logic module, tactical signal and power distribution
Separation system	0.50		0.50		Separation system and ordnance initiation lines not include on space-based KV systems
Ordnance inititiate lines	0.25		0.25		
KV primary battery	1.90	1.80	1.80	1.80	Estimate based on other programs
KV basic structure & installation hardware	5.00	____	3.50	3.50	Kevlar epoxy composite structure used for hign axial and lateral accelerations;propellant tanks used as load carrying structure
Subtotal, KV dry weight less tankage	**44.20**	**22.80**	**34.65**	**33.54**	
Propellant					
Useful Fuel and Oxidizer	60.10	27.10	47.00	45.50	Useful propellant needed to produce a total ΔV of 2000 m/sec
ACS & press fraction of useful fuel	5.0%	n/a	5.0%	5.0%	
ACS & pressurization fuel 5%	3.01		2.35	2.28	Added ACS fuel @5% of divert
Unusable Propellant fraction		n/a	3.0%	3.0%	
Unusable Propellant	1.80	____	1.41	1.37	Propellant trapped in system assumed to be 3% of total
Subtotal, Propellant	63.11	27.10	49.35	47.78	Total propellant includes useful propellant and ACS propellant, of which the amount shown as unusable is trapped in the system.
Tankage	12.62	3.12	9.68	9.37	To calculate tankage mass, LLNL uses 0.115*propellant mass, which is based on a pumped DACS and lower pressure tanks. We assume conventional high pressure tanks with mass 0.2*propellant mass
Total, KV wet	**119.93**	**53.02**	**93.68**	**90.69**	Total KV wet mass includes the dry mass, propellant mass , and tankage mass
ΔV check					
Isp of propellent (sec)	300	300	300	300	
Isp (effective) after ACS & press.	285	285	285	285	To account for ACS/pressure fuel used but not effective for thrust
ΔV from the rocket equation (check)	1999	1999	2002	2002	
ΔV desired (input)	2000	2000	2000	2000	
Endgame acceleration calculation					
KV Mass with 15% fuel remaining	66.29	29.98	51.73	50.08	
Thrust for15g's @ 15% fuel load(Newtons)	9744	4407	7604	7362	Used to estimate thruster size
G's at full fuel load	8.29	8.48	8.28	8.28	

Figure IV-14 Properties of terrestrial- and space-based kill vehicles. Total divert: 2.5 km/s. (From [23])

Finally, the successful interception should end with destruction of the target. This can be achieved by a kinetic kill, which requires a “hit-to-kill.” In “hit-to-kill”, miss distance must be substantially less than the smallest dimension of the target [63]. The conclusion given in [63] indicates that a “hit-to-kill” homing accuracy against highly maneuvering targets in an environment of noisy measurement is hardly feasible. Because the feasibility of the direct hit is so small, we should consider a different approach to destroy the target. In [64], other methods of destruction are described. Lethality enhancement devices (LED), such as hexagon tungsten rods, should be investigated to improve the overall lethality of the KV.

4. The Final Simulation with KVs

The scenario for which we run our simulation is described in Chapter I. Here, we discuss with how the KV is implemented in the simulation.

The target and interceptor are ICBMs as described in [5]. Every ICBM in the simulation is assumed to have four stages that include three booster stages and a final stage that is a warhead for the target and a KV for the interceptor.

The parameters for the ICBM are created in a data matrix. Table IV-8 shows the format of the input data matrix for the generic ICBM used in the simulation.

Table IV-8 The format of the input data matrix.

Stage-#1	**Stage-#2**	**Stage-#3**	**Stage-#4**
Stage weight in lbs, m_s	Stage weight in lbs, m_s	Stage weight in lbs, m_s	Stage weight in lbs, m_s
Stage Fuel in lbs, m_F	Stage Fuel in lbs, m_F	Stage Fuel in lbs, m_F	Stage Fuel in lbs, m_F
Specific Impulse, I_{sp}	Specific Impulse, I_{sp}	Specific Impulse, I_{sp}	Specific Impulse, I_{sp}
Stage burn time in s, t_s	Stage burn time in s, t_s	Stage burn time in s, t_s	Stage burn time in s, t_s

The input data matrix, then, is reintroduced by the function *reformDataMatrix* (see Appendix B) to be able to use it in the simulation. After the reformulation, the data matrix will be in the form of a 8×4 matrix. Table IV-9 shows the format of the final data matrix. The input matrix can be altered to change the parameter of the missile. Note that this simulation is not intended to investigate the optimum parameters for the missiles.

Table IV-9 The format of data matrix that the simulation uses

Stage-#1	**Stage-#2**	**Stage-#3**	**Stage-#4**
Stage weight in kg, m_s	Stage weight in kg, m_s	Stage weight in kg, m_s	Stage weight in kg, m_s
Stage Fuel in kg, m_F	Stage Fuel in kg, m_F	Stage Fuel in kg, m_F	Stage Fuel in kg, m_F
Specific Impulse in s, I_{sp}	Specific Impulse in s, I_{sp}	Specific Impulse in s, I_{sp}	Specific Impulse in s, I_{sp}
Stage burn time in s, t_s	Stage burn time in s, t_s	Stage burn time in s, t_s	Stage burn time in s, t_s
Fuel burn ratio, dm/dt	Fuel burn ratio, dm/dt	Fuel burn ratio, dm/dt	Fuel burn ratio, dm/dt
Canister weight in kg	Canister weight in kg	Canister weight in kg	Canister weight in kg
Next Stage time in s	Next stage time in s	Next stage time in s	Next stage time in s
Total weight in kg	Total weight except first stage in kg	Total weight except first and second stages in kg	Weight of final stage in kg

The stage index moves along the columns of the data matrix (look-up table) while the target properties are along the rows of the data matrix.

The stage change is done by increasing the stage index when the stage-change time comes. The KV, the last stage of the interceptor, is assumed to be launched when the stage index reaches the final value and/or the optimum position is reached. The optimum position is the coordinate in the Cartesian coordinate system that is the best place to launch the KV; then the EKV navigates through the target and gets as close as possible to target. Owing to an unknown target-launch area and the trajectory for the aimpoint, the optimum point can be reached before the interceptor burn-out. Hence, the kill vehicle can

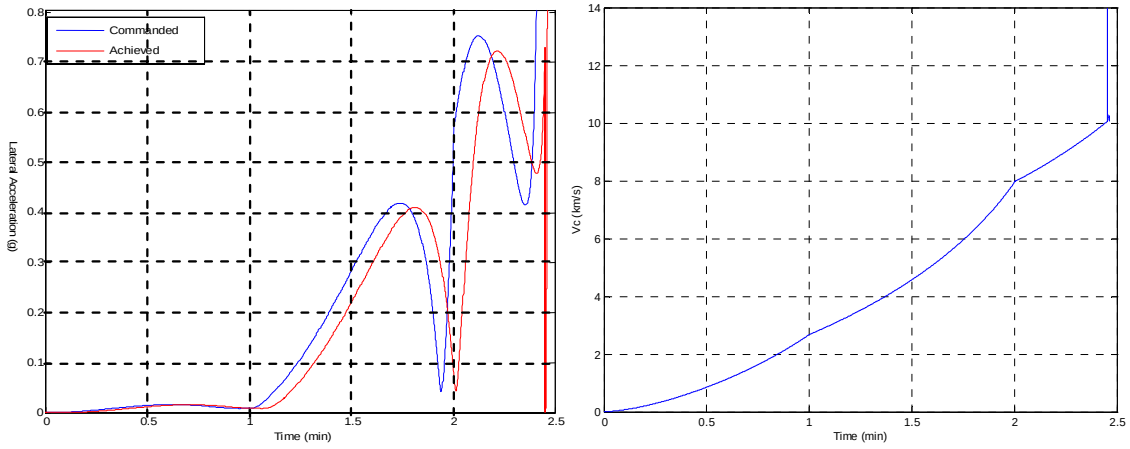
have less velocity than expected. To overcome this effect, we assume that the kill vehicle has enough fuel to burn to increase its speed or maneuverebility.

The optimum point for the KV launch depends on the distance, which is the magnitude of the line-of-sight (LOS) vector between the target and the GBI. We choose a 5-km distance so that we could simulate the "endgame" and gain in the run-time of the simulation, since the simulation time step is decreased down to 500 μs when the KV is launched.

Both the interceptor and the KV are assumed to use PN, which is described in [30]. Because PN is not designed for an accelerating target and the algorithm does not account for the target's expected acceleration, a miss distance is inevitable. In addition to this, since we cannot decrease the simulation time step low enough, we cannot expect to see a miss distance less than 3 m due to the final speed of the target. The target can move 3 m within the reduced time step with a speed of approximately 6,000 km/s.

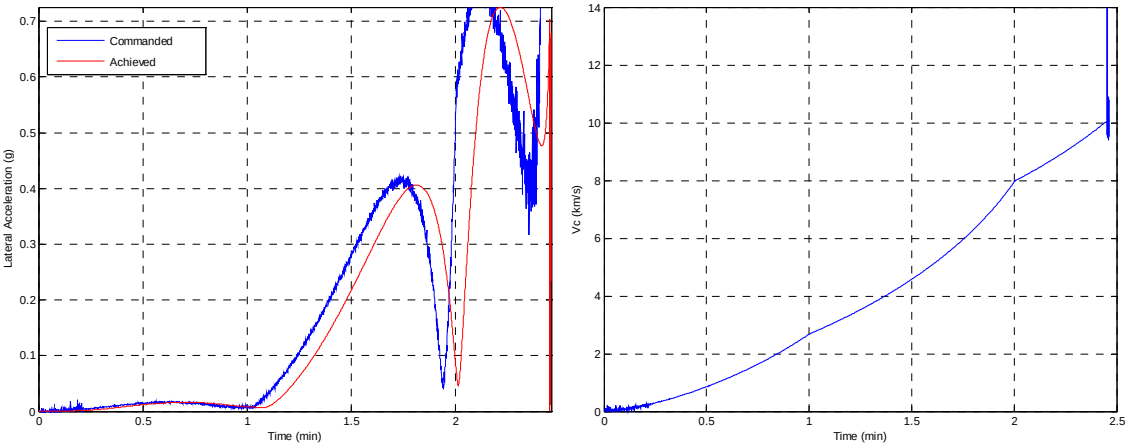
Because the guidance of the interceptor and the kill vehicle is not perfect and contains lags (defined in the simulation as 5 s), the commanded lateral acceleration and the achieved lateral acceleration are not equal.The simulation uses a 3rd order transfer function for the flight control system. This causes an increasing miss distance, even if the tracked position is perfect. Figure IV-15 shows the command lateral acceleration and the closing velocity,V_c, that is reached, with an assumption that the fusion center always knows the true position of the target. The resulting miss distance is about 5 m for both targets.

While tracking errors are given in the MTT section, the guidance commands for a no-EA case and a combined-attack case are given in Figure IV-16 and Figure IV-17. The resulting miss distances on average are 5.8 m and 6.2 m, respectively. Note that these results are for a no-delay launch of the interceptor (launch time for both targets and interceptors are assumed to be the same).



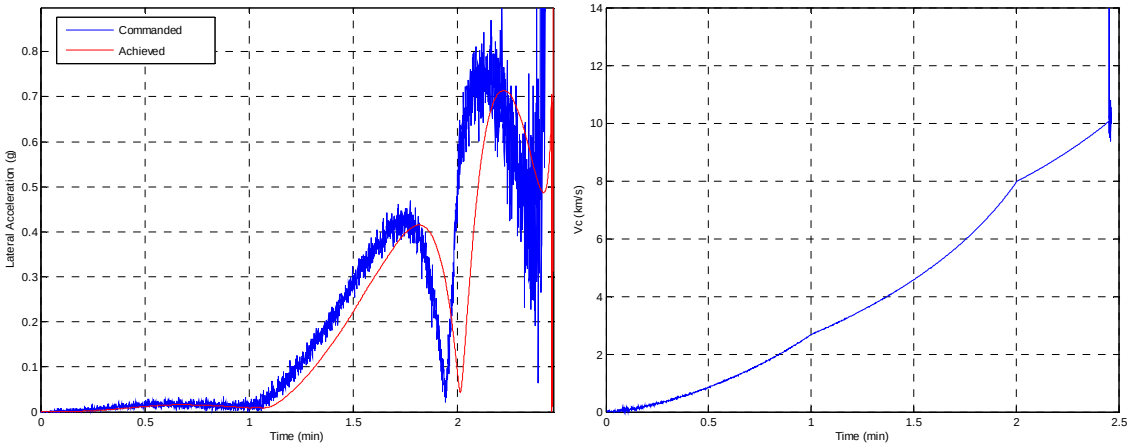
(a) (b)

Figure IV-15 The perfect tracking case (a) Guidance and (b) Closing velocity for MTT algorithm. Zero-delay launch.



(a) (b)

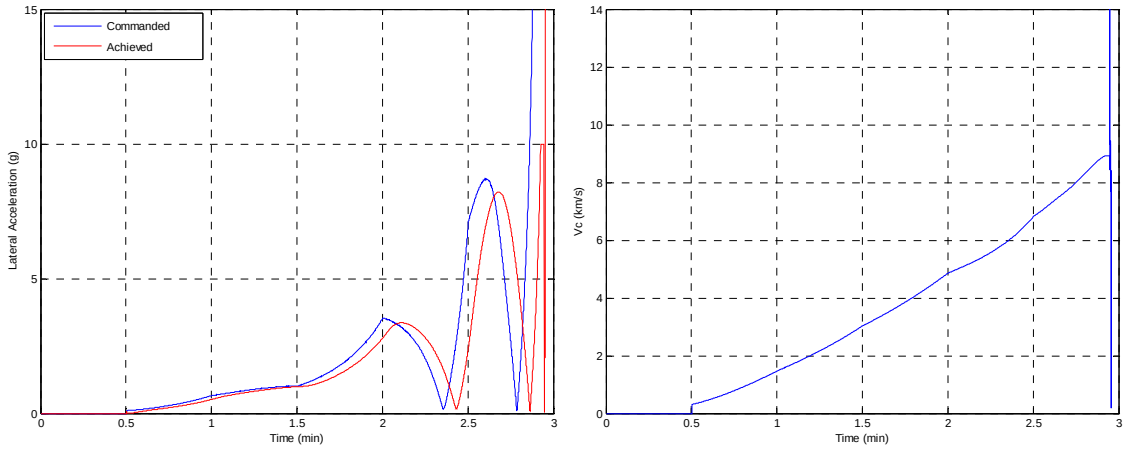
Figure IV-16 The no-EA case (a) Guidance and (b) Closing velocity for MTT algorithm. Zero-delay launch.



(a) (b)

Figure IV-17 Combined EA case (a) Guidance and (b) Closing velocity for MTT algorithm. Zero-delay launch.

A zero-delay-launch is unrealistic since a decision time is needed for launching an interceptor. Furthermore, a track initiation needs to be processed first. For Because of that, the interceptor launch time is increased to 30 s. While the guidance and the Closing velocity have not changed significantly, the miss distance increased to 800 m. Figure IV-18 shows the 30-second-delay launch's guidance and Closing velocity.



(a) (b)

Figure IV-18 The normal case (a) Guidance and (b) Closing velocity for MTT algorithm. 30 second-delay launch.

To obtain an acceptable miss distance, the only parameter we can alter is the interceptor data matrix. Using an exhaustive search, the new stage-burn times are 54 s for every stage, and a 500-lb KV weight is used. The resulting miss distance at best is 10 m. The lateral acceleration, Closing velocity, and resulting lateral divert with the 35-s-delay launch is shown in Figure IV-19 through Figure IV-21.

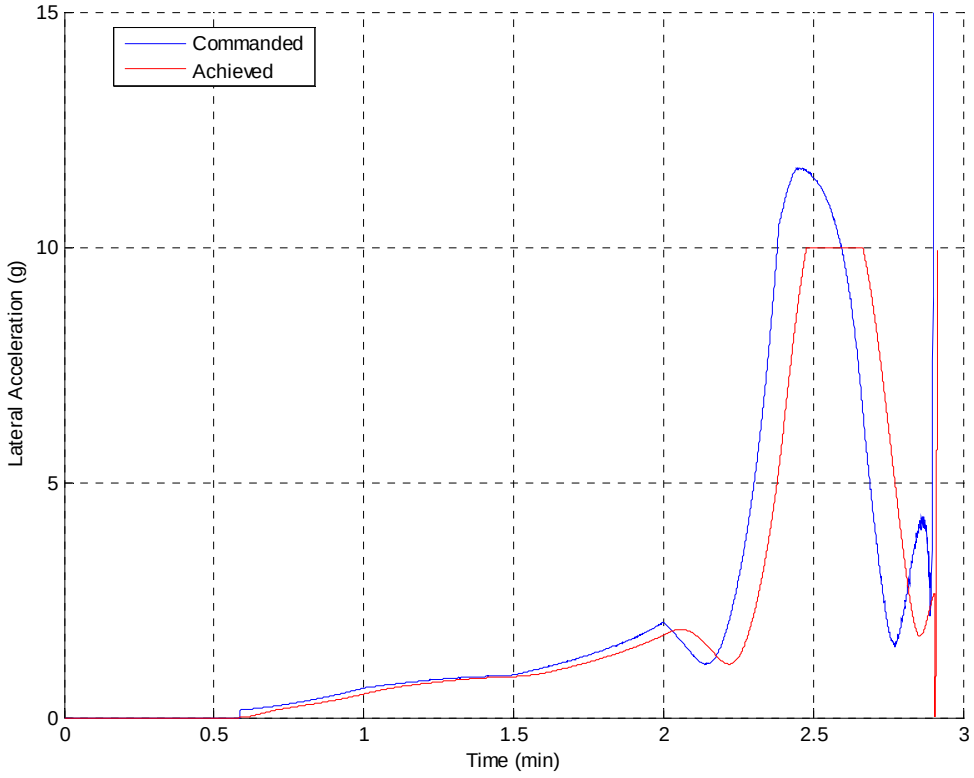


Figure IV-19 The no-EA case lateral acceleration for the MTT algorithm with a 35-second-delay launch.

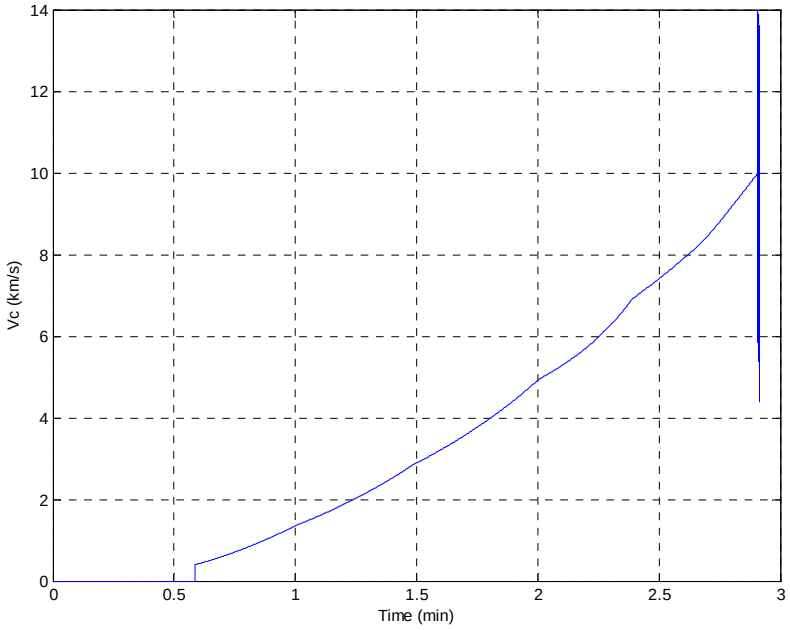


Figure IV-20 The no-EA case closing velocity for the MTT algorithm with a 35-second-delay launch

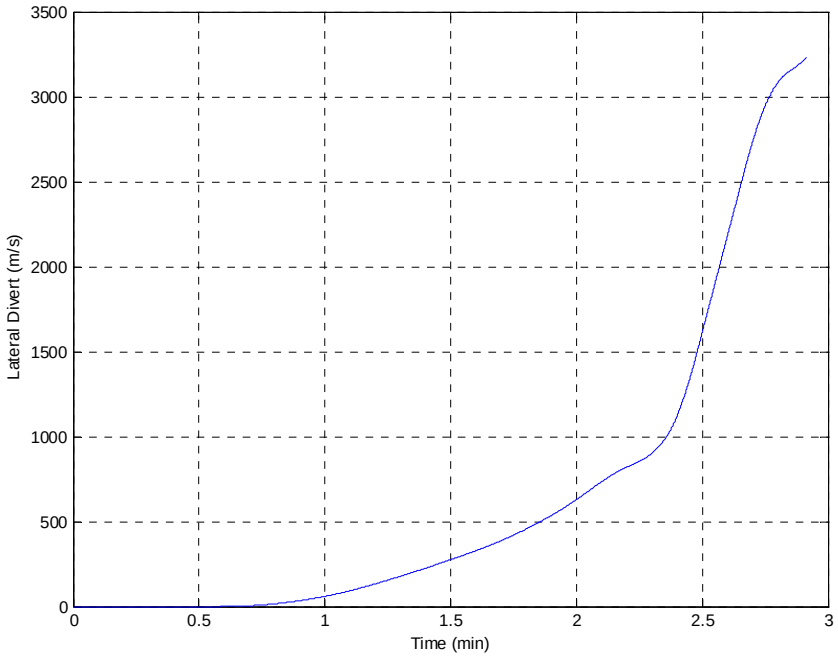


Figure IV-21 The no-EA case Lateral Divert for the MTT algorithm with a 35-second-delay launch

The resulting interception velocity plot for the target and the interceptor is shown in Figure IV-22.

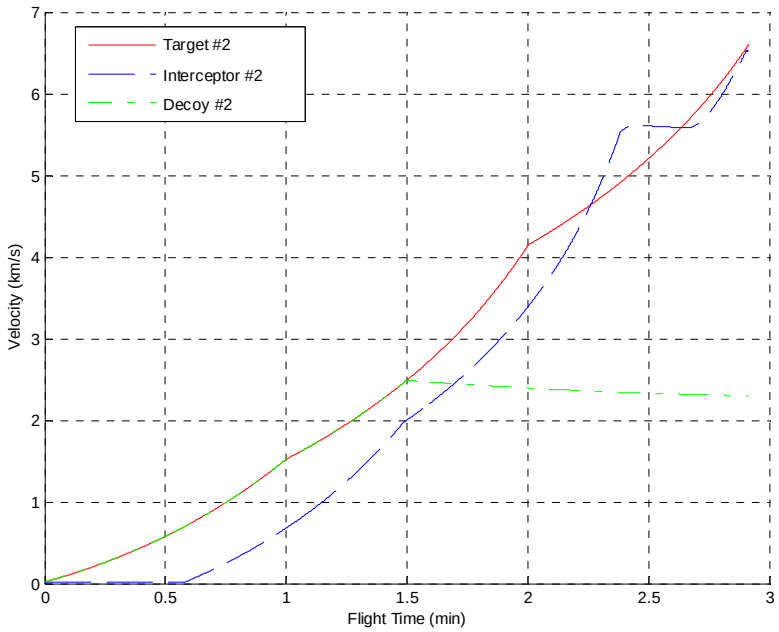


Figure IV-22 35 s launch delay Velocities

The miss distance can be enhanced by approaching the target launch site, but this causes an initial location distance of approximately 234 km, which is unlikely. For evaluation of the delayed launch time, we keep the interception time within the targets' boost time. For the 35-s delayed launch, the interception takes place at 172 s.

C. SUMMARY

In this section, we investigate the basic requirements for the MTT algorithm and the KV. Target discrimination and initialization are addressed; they are assumed to be aided by the IR sensors. The correlation is implemented using ellipsoidal gating; the association is realized using a PDAF-type sub-optimal Bayesian algorithm. For these purposes, two new functions are developed that work concurrently. The fusion is implemented using an inverse of the trace of the state covariance matrix within the *mesh.m* (see Appendix B) function.

For the KV, the evolution is summarized. The type and specification of the on-board sensor are given. The kill requirement is explained and the results of the KV simulation are explained. The best results for the delayed launch are acquired when we use decreased burn time.

THIS PAGE INTENTIONALLY LEFT BLANK

V. CONCLUSION

A. SUMMARY OF THE WORK

In this thesis, we investigated the effect of EAs on the RF sensors within a boost-phase ballistic missile defense system. The methods to alleviate these attacks in a multi-target, multi-sensor scenario were also addressed.

The EA types that might be used against the RF sensors of the boost-phase ballistic missile defense system were presented. The RCS reduction, the chaff, deceptive jamming, and expendable decoys were described in detail. We used simulations to show that RCS reduction and deceptive jamming are the major methods that degrade the performance of the interceptor, while chaff and decoys are not as practical.

Because the EA effect is so intense that we investigated a new way to mitigate it. A fusion algorithm based on Kalman filter was developed to minimize the effect. Kalman-based and the Bayesian-based fusion algorithms were compared. Although the proposed fusion algorithm can easily deal with individual EA techniques, a combined attack of all considered EA methods in a sequential way degraded the performance significantly.

The MTT and the KV were studied using the simulation. A multi-target multi-sensor environment was simulated, using two target/interceptor pairs and three RF sensors. Correlation was implemented with an ellipsoidal gating, and association was realized using a probabilistic data association type algorithm. The nonlinear triangulation was resolved using a nonlinear measurement function. The kill vehicle endgame was simulated by altering the data matrix of the interceptor. A delayed-time interceptor was also simulated. A delayed launch of the interceptor or use of the PN algorithm increases the miss distance, even if the tracking is perfect. To enhance the interception performance, the weight of the KV was reduced to 250 kg and the interceptor burn-out time was reduced to162 s. This resulted in a 172-s interception performance with a 35-s-delayed-launch, just 8 s less than the target burn-out time, and a 10-m miss distance.

B. SIGNIFICANT RESULTS

We have used a 3D multi-target, multi-sensor simulation to evaluate the boost-phase ballistic missile defense system. The following significant results were obtained.

The RF sensor is very vulnerable to any EA that might be used. Deception jamming forces the radar to use angle-only measurements while the RCS reduction increases the standard deviation of the measurement error. Decoys and the chaff lure the radar toward them, but are not practical.

The MTT algorithm has the capacity to eliminate EA effects if designed properly. If the RF sensor probability of detection is not equal to unity, the MTT algorithm loses the track, since the target in question has a high acceleration capacity.

The simulation results show that the closest the KV can get to the ICBM target is 5 m for all cases. Because a hit-to-kill requires a miss distance of less than the smallest dimension of the target (2.6 m in the simulation), the hit-to-kill approach is not applicable to the boost-phase ballistic missile defense system, which may be attributed to the high target acceleration.

C. SUGGESTIONS FOR FUTURE WORK

In this work, a multi-target, multi-sensor scenario was investigated. The considered sensors were all the same type. Because of this, target discrimination and track initialization were addressed but not implemented in the simulation. In a future study, the passive IR sensor, discrimination and initiation issues may be investigated.

In this thesis, the KV was investigated only for the endgame stage. In a future study, a full-scale KV model with a more advanced guidance algorithm may be developed. For this study, we worked with the point mass as the targets and the interceptors. An extended body version should be implemented in a future effort.

APPENDIX A. CODE FLOWCHART

This appendix includes the flowchart of the main simulation of the MATLAB® code. The flowchart can be used with the code listed in Appendix B and "Read Me" in Appendix C to understand and modify the code for future use.

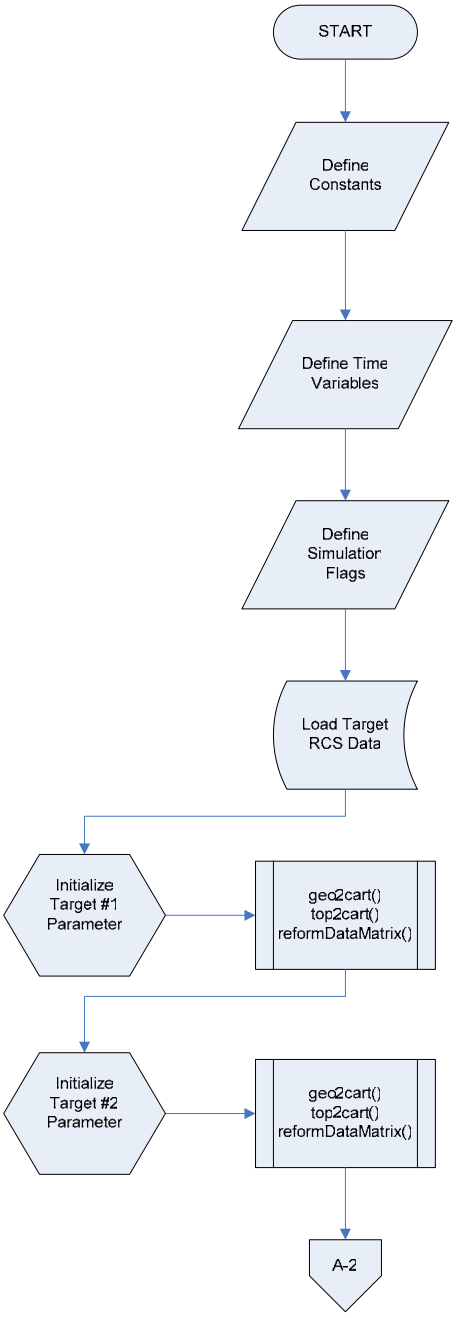


Figure A-1 Flowchart-- data start-up (1 of 8)

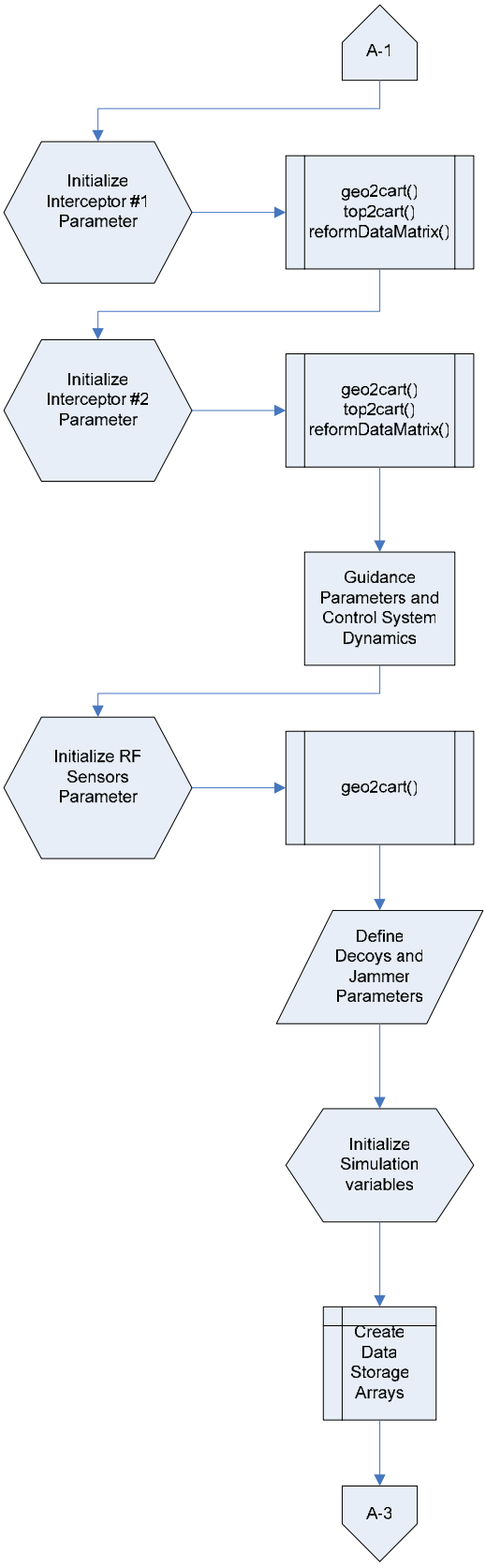


Figure A-2 Flowchart-- data start-up cont' (2 of 8)

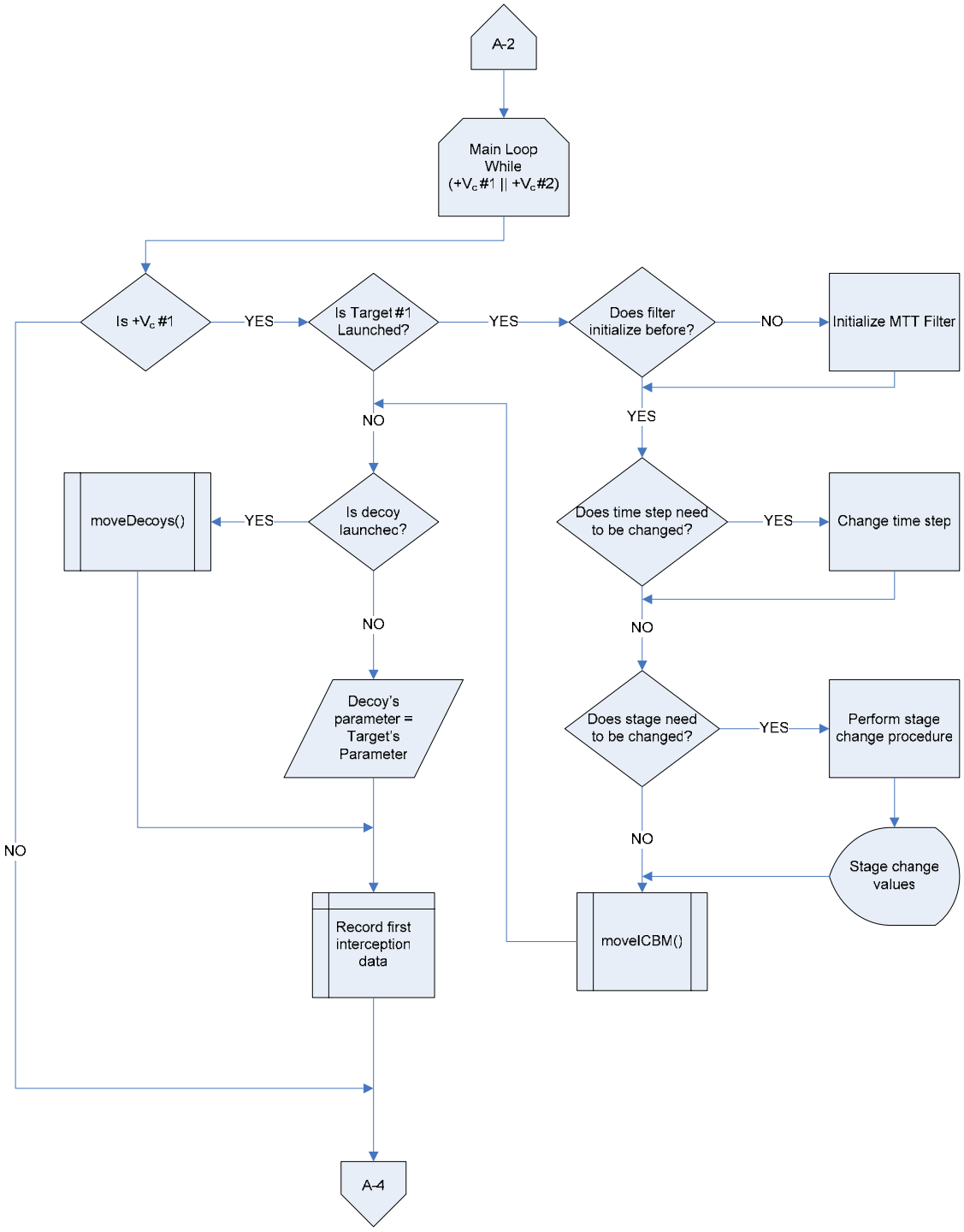


Figure A-3 Flowchart-- first target motion (3 of 8)

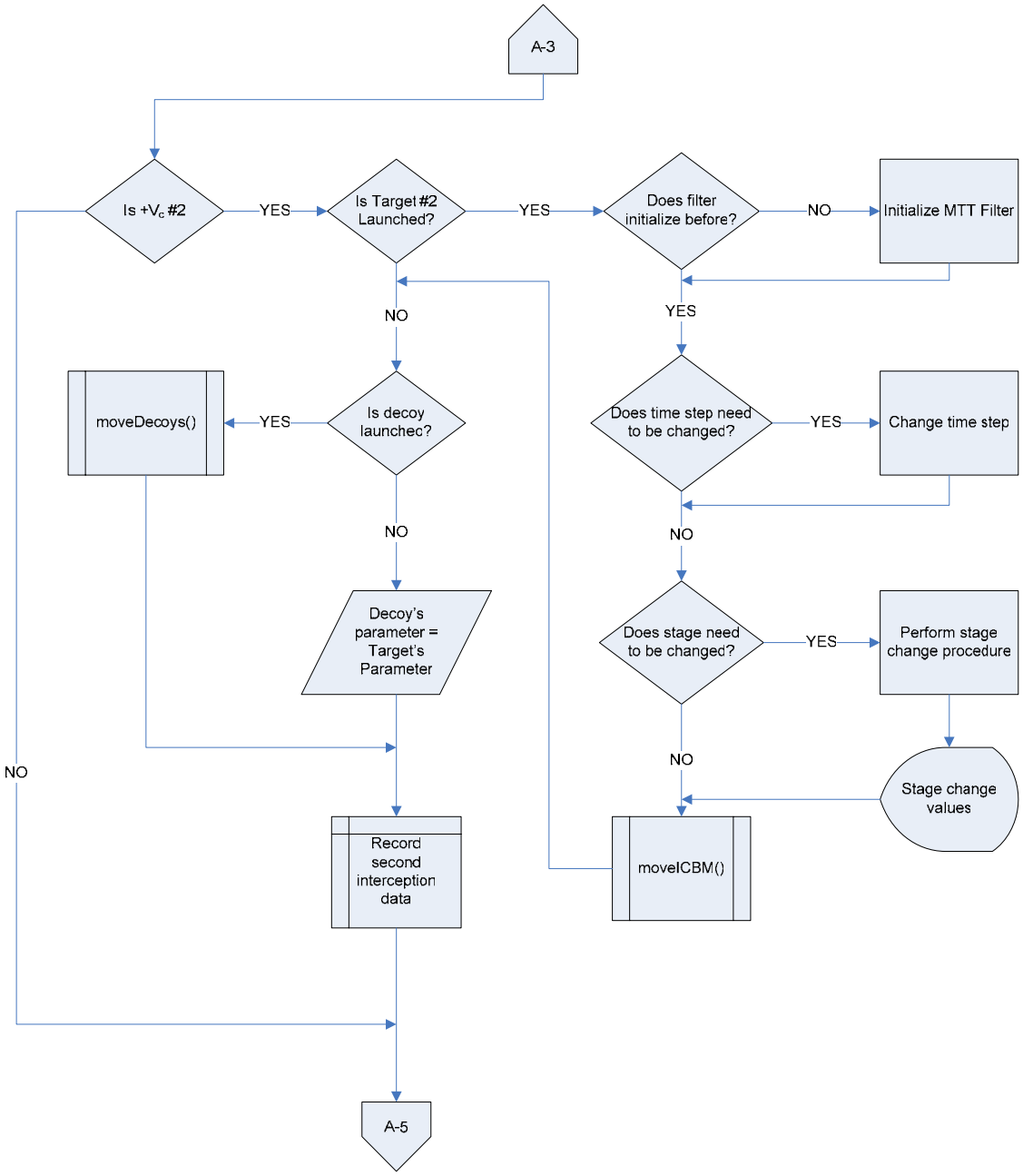


Figure A-4 Flowchart-- second target motion (4 of 8)

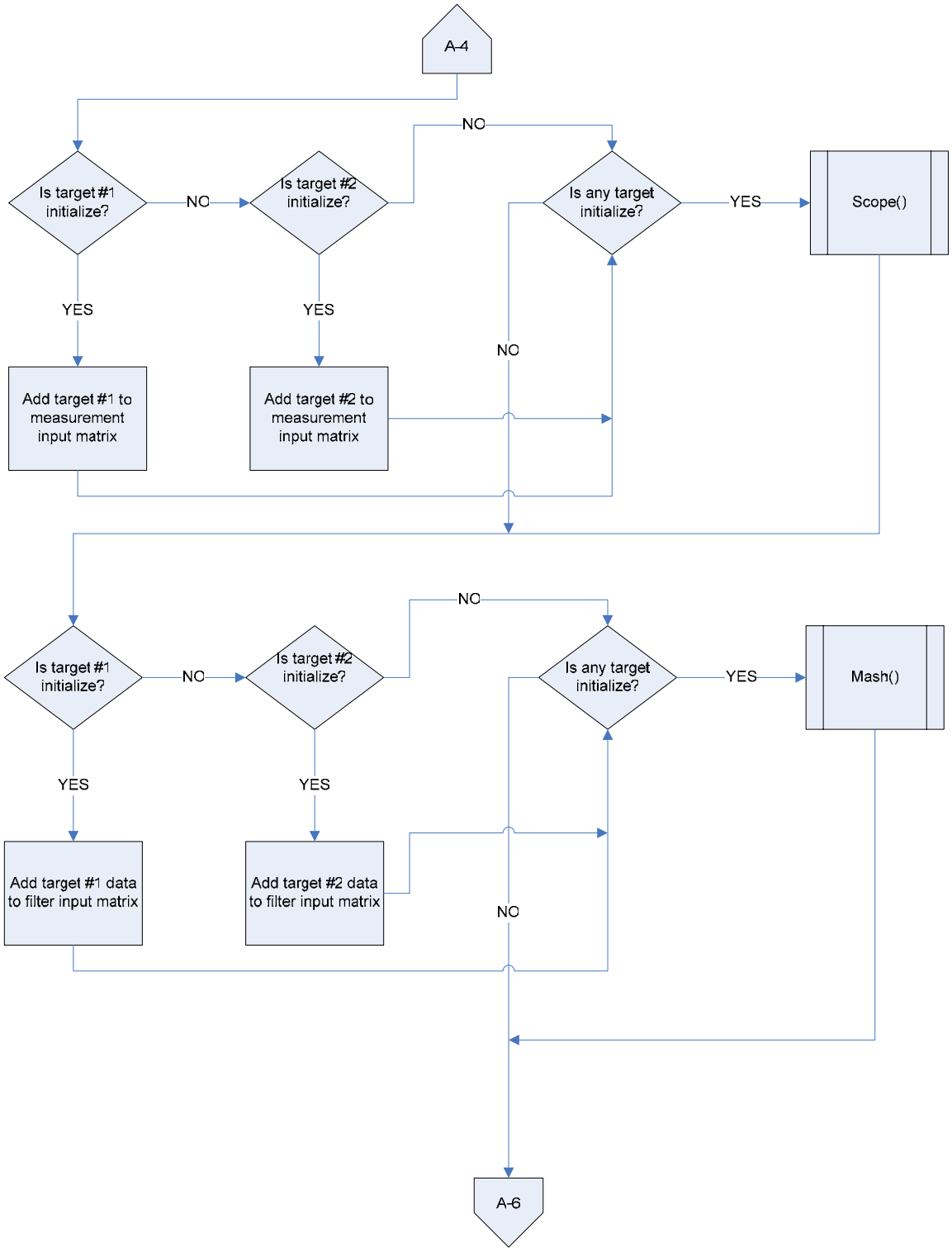


Figure A-5 Flowchart-- measurement and MTT (5 of 8)

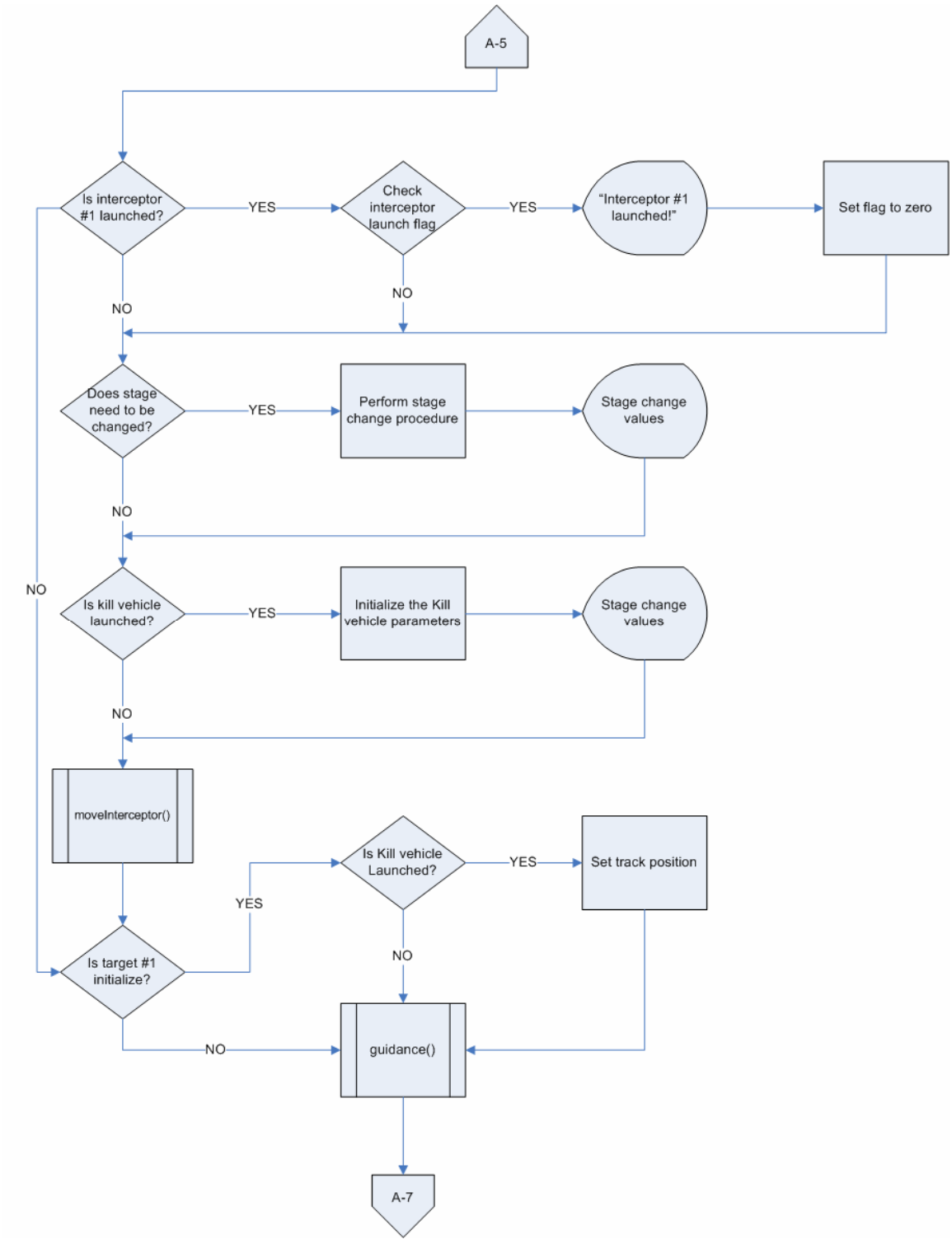


Figure A-6 Flowchart-- first interceptor motion (6 of 8)

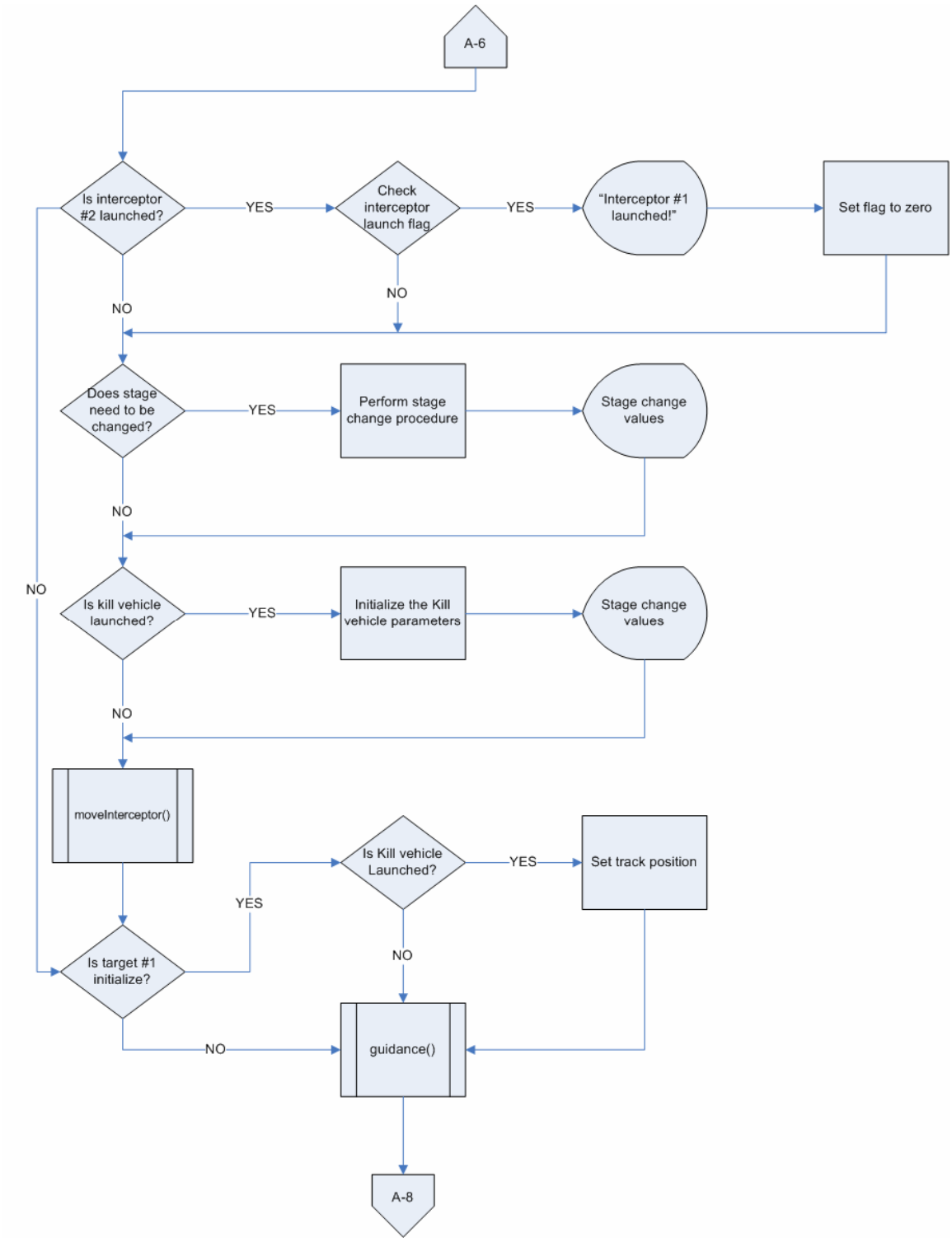


Figure A-7 Flowchart-- second interceptor motion (7 of 8)

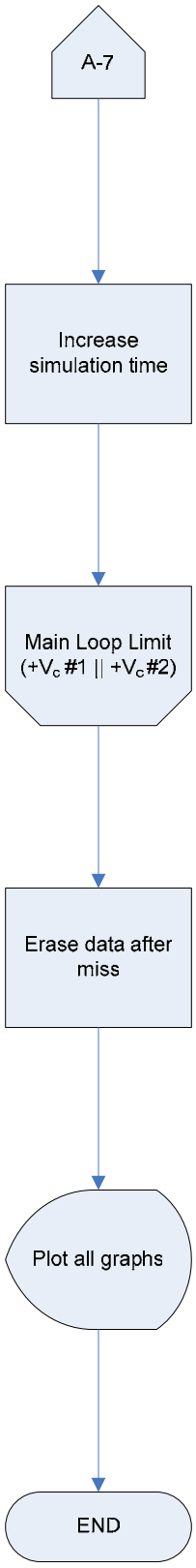


Figure A-8 Flowchart-- finalize the simulation (8 of 8)

APPENDIX B. MATLAB® CODE

This appendix includes a full copy of the simulation and the subroutines of the MATLAB® code. All subroutines are written in order of appearance.

A. MULTITARGET3D ( )- (MAIN SIMULATION)

```
% MultiTarget3D Simulation for
% Multi-Target Multi-Sensor Multi-stage Boosting Boost-phase Interception
% APR 2005, Monterey, CA
% K.Yildiz, Prof. P.E. Pace, Prof. M.Tummala

clear;clc;
tic;            % Calculate Run time
global RCS1X RCS1X_R txDelay  maxG navCoefM SSM1 SSM2 stateM1 stateM2 timeFlags...
       Vcfirst updateTime;

%-------------------------------------------------------------------------------
%% Constants
Re = 6.37e6;              % Radius of the Earth (m)
G = 6.67e-11;             % Gravitational Constant (m^3/s^2.kg)
Me = 5.98e24;             % Earth's Mass (kg)
SOL = 299792458;          % Speed of light (m/s)
posEC = [0; 0; 0];        % Position of Earth's Center
SMALL = -1e-6;            % An arbitrary small number for comparisons
BIG = 1e6;                % An arbitrary big number for comparisons
Hp = [1 0 0 0 0 0;
      0 1 0 0 0 0;
      0 0 1 0 0 0];       % The position observation matrix
Hv = [0 0 0 1 0 0;
      0 0 0 0 1 0;
      0 0 0 0 0 1];       % velocity observation matrix

%% Time variable
farTimeStep = 0.05;       % Integration Time Step at Interception Phase(s)
nearTimeStep = 0.0005;    % Integration Time Step at Terminal Phase(s)
timeStepSwitch = 5000;    % Distance to Switch Time Step (m)
t1 = 0;                   % Independent first Interception Time (s)
t2 = 0;                   % Independent second Interception Time (s)
t = [t1,t2];              % Simulation Time (s)
dt1 = 0.05;               % First interception time step (s)
dt2 = 0.05;               % Second interception time step (s)
dt = [dt1,dt2];           % Simulation Time Step (s)
updateTime_1 =...
    farTimeStep*1;        % Sensor Update Interval (s)
updateTime_2 =...
    farTimeStep*1;        % Sensor Update Interval (s)
updateTime = [updateTime_1, updateTime_2];
txDelay1 = 10e-3;         % First interception Transmission delay (s)
txDelay2 = 10e-3;         % Second interception Transmission delay (s)
txDelay = [txDelay1, txDelay1];
txCounter_1 = 0;          % First interception data Update counter
txCounter_2 = 0;          % Second interception data Update counter
txCounter = [txCounter_1, txCounter_2];

%% Flags
kill_launch = [1 1];      % Launching Kill vehicle
initializing = [1 1];     % JPDA Filter Initialization flags
isDecoys = [0 0];         % Decoys launch flag
isReduced = 0;            % Reduced RCS indication flag
isJamming = 0;            % Jamming indication flag
timeStepFlag = [1 1];     % Flag used to switch between time steps
launch_flag = [1 1];      % For indicating the Interceptor launch
Vcfirst = [1 1];          % Flag for setting the First Vc to zero
txFlag_1 = 1;             % First interception Data Update flag
txFlag_2 = 1;             % Second interception Data Update flag
```

```
txFlag = [1, 1];
timeFlags =  [t;
              dt;
              txFlag;
              txCounter;
              txDelay];  % Put All time flag into one

%% Initialization of Parameters
% Load Target RCS Data Use the X band radar
load POstage1_X; RCS1XRaw = Sth;    % Stage-1, X-Band
load RCS1X_R; RCS1X_RRaw = Sth;     % Stage-1, Reduced
% Interpolate RCS Data for 0.1 degrees precision
mAngleDegRaw = 0:360;        % Monostatic Angle Theta at Raw Data Precision
mAngleDeg = 0:0.1:360;       % Monostatic Angle Theta after Interpolation
RCS1X = interp1(mAngleDegRaw,...
        RCS1XRaw, mAngleDeg);% Generate 0.1 deg precision RCS Matrix
RCS1X_R = interp1(mAngleDegRaw,...
      RCS1X_RRaw, mAngleDeg);% Generate 0.1 deg precision RCS Matrix
%--------------------------------------------------------------------------

% Target- #1
lAzT_1 = deg2rad(50.1);      % Target Launch Angle (Azimuth) (Radians)
lElT_1 = deg2rad(84);        % Target Launch Angle (Elevation) (Radians)

% Target (Located at Kilju Missile Base, N. Korea)
% Position the target in Cartesian Coordinates.
% The Target located at (N41'00" / E129'00")
posT_1 = geo2cart('41d00m00sN','129d00m00sE',Re); % Target Position Vector
unitvT_1 = top2cart(lAzT_1, lElT_1,...
                    '41d00m00sN', '129d00m00sE'); % Velocity Unit Vector
pos0T_1 = posT_1;                                 % Target Initial Position
lTimeT_1 = 0;                                     % Target Launch Time (s)
lenT_1 = 21.8;                                    % Target Total Length (m)

%Define Missile Data Matrices as Follows
%                           Stage-1     Stage-2     ...     Stage-n
%   Total Mass (lb)       [ X           X           ...     X
%   Propellent Mass (lb)    X           X           ...     X
%   Specific Impulse (s)    X           X           ...     X
%   In-stage Burn Time (s)  X           X           ...     X ]
%Following Rows are Added by the Program (Do not Define!)
%   dM/dt                 [ X           X           ...     X
%   Canister Mass (kg)      X           X           ...     X
%   Ignition Time           X           X           ...     X
%   Total Mass (kg)         X           X                   X ]

%Generic 3-Stage Missile Data Matrix
%               Stage-1     Stage-2     Stage-3     Payload
dataMatrixT_1 = [ 108000      61000       17000       5000
                  91800       51850       14450       0
                  300         300         300         0
                  60          60          60          1   ];
%Add dM/dt and Canister Weight Rows to Data Matrix
dataMatrixT_1 = reformDataMatrix(dataMatrixT_1);
MT_1 = sum(dataMatrixT_1(1,:)); %Target Initial Total Mass
%--------------------------------------------------------------------------

% Target- #2
lAzT_2 = deg2rad(50);    % Target Launch Angle (Azimuth) (Radians)
lElT_2 = deg2rad(84);    % Target Launch Angle (Elevation) (Radians)

% Target (Located at Ok'pyong Missile Base, N. Korea)
% Position the target in Cartesian Coordinates.
% The Target located at (N39'25" / E127'25")
posT_2 = geo2cart('39d25m00sN', '127d25m00sE', Re);% Target Position Vector
pos0T_2 = posT_2;                                  % Target Initial Position
unitvT_2 = top2cart(lAzT_2, lElT_2,...
                    '39d25m00sN', '127d25m00sE'); % Velocity Unit Vector
lTimeT_2 = 0;                                     % Target Launch Time (s)
lenT_2 = 21.8;                                    % Target Total Length (m)
```

```
%Generic 3-Stage Missile Data Matrix
%               Stage-1     Stage-2     Stage-3     Payload
dataMatrixT_2 = [ 108000      61000       17000       5000
                  91800       51850       14450       0
                  300         300         300         0
                  60          60          60          1   ];
%Add dM/dt and Canister Weight Rows to Data Matrix
dataMatrixT_2 = reformDataMatrix(dataMatrixT_2);
MT_2 = sum(dataMatrixT_2(1,:)); % Target Initial Total Mass
% ------------------------------------------------------------------------

% Missile interceptor- #1
lAzM_1 = deg2rad(298);    % Missile Launch Angle (Azimuth) (Radians)
lElM_1 = deg2rad(79);     % Missile Launch Angle (Elevation) (Radians)

%Missile (Located at Sea of Japan, 600km East of Target Launch Site)
% Position the interceptor in Cartesian Coordinates.
% The missile located at (N41'00" / E136'07")
posM_1 = geo2cart('41d00m00sN', '136d07m30sE', Re);% Missile Position Vector
pos0M_1 = posM_1;                                  % Missile Initial Position
unitvM_1 = top2cart(lAzM_1, lElM_1,...
                    '41d00m00sN', '136d07m30sE'); % Velocity Unit Vector
lTimeM_1 = lTimeT_1 + 35;                          % Missile Launch Time (s)
lenM_1 = 21.8;                                     % Missile Total Length (m)

%Generic 3-Stage Interceptor Data Matrix use the Generic Missile GM3
dataMatrixM_1 = [ 108000      61000       17000       500
                  102600      57950       16150       200
                  300         300         300         300
                  54          54          54          4];
%Add dM/dt and Canister Weight Rows to Data Matrix
dataMatrixM_1 = reformDataMatrix(dataMatrixM_1);
MM_1 = sum(dataMatrixM_1(1,:));                % Missile Initial Total Mass
% ------------------------------------------------------------------------

% Missile interceptor- #2
lAzM_2 = deg2rad(298);    % Missile Launch Angle (Azimuth) (Radians)
lElM_2 = deg2rad(79);     % Missile Launch Angle (Elevation) (Radians)

%Missile (Located at Sea of Japan, 600km East of Target Launch Site)
% Position the interceptor in Cartesian Coordinates.
% The missile located at (N39'25" / E134'21")
posM_2 = geo2cart('39d26m00sN', '134d21m30sE', Re);% Missile Position Vector
pos0M_2 = posM_2;                                  % Missile Initial Position
unitvM_2 = top2cart(lAzM_2, lElM_2,...
                    '39d26m00sN', '134d21m30sE');% Velocity Unit Vector
lTimeM_2 = lTimeT_2 + 35;                          % Missile Launch Time (s)
lenM_2 = 21.8;                                     % Missile Total Length (m)

%Generic 3-Stage Interceptor Data Matrix use the Generic Missile GM3
dataMatrixM_2 = [ 108000      61000       17000       500
                  102600      57950       16150       0
                  300         300         300         300
                  54          54          54          4];

%Add dM/dt and Canister Weight Rows to Data Matrix
dataMatrixM_2 = reformDataMatrix(dataMatrixM_2);
MM_2 = sum(dataMatrixM_2(1,:)); %Missile Initial Total Mass
%------------------------------------------------------------------------

% Navigation parameter for the interceptors
navCoefM = [4 4];         % Missile Navigation Coefficient for Proportional
                          % Navigation (Either 3, 4 or 5)
maxG = [10 10];           % Max Lateral Acceleration Command (g)

%Define Control System Dynamics Transfer Function
%                       1
%           -------------------------
%           as^n+ ... + bs^2 + cs + 1
TMc = 5;                                       % System Time Constant
numTFM = 1;                                    % Numerator
```

```
denTFM = [(TMc^3/27) (TMc^2/3) TMc 1];         % Denominator (3rd Order TF)
sysTFM = tf(numTFM, denTFM);   % Generate Transfer Function From Parameters
sysdTFMFar = c2d(sysTFM,farTimeStep);          % Discretize System (Far)
sysdTFMNear = c2d(sysTFM,nearTimeStep);        % Discretize System (Near)
% Convert Transfer Function to State Space (Far)
[AMFar,BMFar,CMFar,DMFar] = ssdata(sysdTFMFar);
% Convert Transfer Function to State Space (Near)
[AMNear,BMNear,CMNear,DMNear] = ssdata(sysdTFMNear);
% Put this matrix in one, to use separate it
SSMfar = [AMFar,BMFar,CMFar',[DMFar;0;0]];
SSMnear = [AMNear,BMNear,CMNear',[DMNear;0;0]];
%--------------------------------------------------------------------------

%Various Computations
g0 = (G * Me) / (Re ^ 2);   %Gravitational Acceleration at Sea Level
%--------------------------------------------------------------------------

% RF-1 (125 degrees 600 km from Target Launch Site, Sea of Japan)
% The RF-1 located at (N37'46" / E134'35")
posRF1 = geo2cart('40d21m00sN', '134d34m35sE', Re);  % RF-1 Position Vector

% RF-2 (135 degrees 600 km from Target Launch Site, Sea of Japan)
% The RF-2 located at (N37'05" / E133'46")
posRF2 = geo2cart('43d34m00sN', '135d46m00sE', Re);  % RF-2 Position Vector

%RF-3 ( Sea of Japan)
% The RF-3 located at (N39'35" / E130'46")
posRF3 = geo2cart('39d35m00sN', '130d46m00sE', Re);  % RF-3 Position Vector
%--------------------------------------------------------------------------

% Decoy for Target- #1
posD_1 = posT_1;                               % Decoy Position Vector
lTimeD_1 = lTimeT_1 + 90;                      % Decoy Release Time (s)
unitvD_1 = unitvT_1;                           % Velocity Unit Vector

% Decoy for Target- #2
posD_2 = posT_2;                               % Decoy Position Vector
lTimeD_2 = lTimeT_2 + 90;                      % Decoy Release Time (s)
unitvD_2 = unitvT_2;                           % Velocity Unit Vector
% --------------------------------------------------------------------------

% Jammer on the Target-1 and Target-2
% Position = Current Position of target
PtJ = 1e3  ;                                   % Jammer Power (W)
deltaFJ = 4e9;                                 % Jammer Bandwidth (Hz)
%--------------------------------------------------------------------------

%% Simulation Start the variable
disp(' ');                                % Display a blank line
%Target-1
stageT_1 = 1;                             % Target Stage
stageChangeTimeT_1 = ...
    dataMatrixT_1(7, stageT_1 + 1);       % Target Next Stage Change Time (s)
ISPT_1 = dataMatrixT_1(3, stageT_1);      % Target Stage Specific Impulse (s)
dMdtT_1 = dataMatrixT_1(5, stageT_1);     % Target Stage dM/dT (kg/s)
hT_1 = 0;                                 % Target Height (m)
aT_1 = [0; 0; 0];                         % Target Acceleration Vector
magTT_1 = dMdtT_1 * g0 * ISPT_1;          % Target Stage Sea Level Thrust (N)
magvT_1 = (magTT_1 - MT_1 * g0) /...
    MT_1 * (lenT_1 * MT_1/...
    (magTT_1 - MT_1 * g0)) ^ (1 / 2);     % Target Silo Exit Velocity (m/s)
vT_1 = magvT_1 .* unitvT_1;               % Target Velocity Vector
groundTrackT_1 = posT_1;                  % Target Ground Track Vector
oldGroundTrackT_1 = groundTrackT_1;       % Target Old Ground Track
distT_1 = 0;                              % Target Ground Distance (m)
mnvrT_1 = 0;                              % Target Maneuver (g)
sensedPosT_1 = posT_1;                    % Target Sensed Position Vector
poserror_1 = 0;                           % Initial position error
xT_1 = [posT_1;vT_1];                     % State vector of ICBM

%Target-2
```

```
stageT_2 = 1;                           % Target Stage
stageChangeTimeT_2 =...
    dataMatrixT_2(7, stageT_2 + 1);     % Target Next Stage Change Time (s)
ISPT_2 = dataMatrixT_2(3, stageT_2);    % Target Stage Specific Impulse (s)
dMdtT_2 = dataMatrixT_2(5, stageT_2);   % Target Stage dM/dT (kg/s)
hT_2 = 0;                               % Target Height (m)
aT_2 = [0; 0; 0];                       % Target Acceleration Vector
magTT_2 = dMdtT_2 * g0 * ISPT_2;        % Target Stage Sea Level Thrust (N)
magvT_2 = (magTT_2 - MT_2 * g0) /...
    MT_2 * (lenT_2 * MT_2/...
    (magTT_2 - MT_2 * g0)) ^ (1 / 2);   % Target Silo Exit Velocity (m/s)
vT_2 = magvT_2 .* unitvT_2;             % Target Velocity Vector
groundTrackT_2 = posT_2;                % Target Ground Track Vector
oldGroundTrackT_2 = groundTrackT_2;     % Target Old Ground Track
distT_2 = 0;                            % Target Ground Distance (m)
mnvrT_2 = 0;                            % Target Maneuver (g)
sensedPosT_2 = posT_2;                  % Target Sensed Position Vector
poserror_2 = 0;                         % Initial position error
xT_2 = [posT_2;vT_2];                   % State-Space matrix of ICBM
%-------------------------------------------------------------------------

%Missile interceptor- #1
stageM_1 = 1;                           % Missile Stage
stageChangeTimeM_1 =...
    dataMatrixM_1(7, stageM_1 + 1);     % Missile Next Stage Change Time(s)
ISPM_1 = dataMatrixM_1(3, stageM_1);    % Missile Stage Specific Impulse(s)
dMdtM_1 = dataMatrixM_1(5, stageM_1);   % Missile Stage dM/dT (kg/s)
hM_1 = 0;                               % Missile Height (m)
aM_1 = [0; 0; 0];                       % Missile Acceleration Vector
gM_1 = g0;                              % Missile gravitational Acc.
magTM_1 = dMdtM_1 * g0 * ISPM_1;        % Missile Stage Sea Level Thrust (N)
magvM_1 = (magTM_1 - MM_1 * g0) /...
    MM_1 * (lenM_1 * MM_1/...
    (magTM_1 - MM_1 * g0)) ^ (1 / 2);   % Missile Silo Exit Velocity (m/s)
vM_1 = magvM_1 .* unitvM_1;             % Missile Velocity Vector
groundTrackM_1 = posM_1;                % Missile Ground Track Vector
oldGroundTrackM_1 = groundTrackM_1;     % Missile initial ground track
distM_1 = 0;                            % Missile Ground Distance (m)
GFM_1 = [0;0;0];                        % Missile Guidance Force (N)
comLatAccM_1 = 0;                       % Commanded Lateral Acceleration (g)
achLatAccM_1 = 0;                       % Achieved Lateral Acceleration (g)
latDivM_1 = 0;                          % Missile Lateral Divert (m/s)
nlM_1 = [0; 0; 0];                      % Achieved Lateral Acceleration
SSM1  = SSMfar;                         % State Space matrix for interceptor #1
stateM1 = [0 0 0;                       % Initial StateMx
           0 0 0;                       % Initial StateMy
           0 0 0];                      % Initial StateMz
magGFM_1 = 0;                           % Magnitude of Guidance Force (N)
magncM_1 = 0;                           % Magnitude of the lat. acc.
xM_1 = [posM_1;vM_1];                    % State-Space matrix of Interceptor

%Missile interceptor-2
stageM_2 = 1;                           % Missile Stage
stageChangeTimeM_2 =...
    dataMatrixM_2(7, stageM_2 + 1);     % Missile Next Stage Change Time(s)
ISPM_2 = dataMatrixM_2(3, stageM_2);    % Missile Stage Specific Impulse(s)
dMdtM_2 = dataMatrixM_2(5, stageM_2);   % Missile Stage dM/dT (kg/s)
hM_2 = 0;                               % Missile Height (m)
aM_2 = [0; 0; 0];                       % Missile Acceleration Vector
gM_2 = g0;                              % Gravitational Acceleration
magTM_2 = dMdtM_2 * g0 * ISPM_2;        % Missile Stage Sea Level Thrust(N)
magvM_2 = (magTM_2 - MM_2 * g0) /...
    MM_2 * (lenM_2 * MM_2/...
    (magTM_2 - MM_2 * g0)) ^ (1 / 2);   % Missile Silo Exit Velocity (m/s)
vM_2 = magvM_2 .* unitvM_2;             % Missile Velocity Vector
groundTrackM_2 = posM_2;                % Missile Ground Track Vector
oldGroundTrackM_2 = groundTrackM_2;     % Missile initial ground track
distM_2 = 0;                            % Missile Ground Distance (m)
GFM_2 = [0;0;0];                        % Missile Guidance Force (N)
comLatAccM_2 = 0;                       % Commanded Lateral Acceleration (g)
achLatAccM_2 = 0;                       % Achieved Lateral Acceleration (g)
```

```
latDivM_2 = 0;                          % Missile Lateral Divert (m/s)
nlM_2 = [0; 0; 0];                      % Achieved Lateral Acceleration
SSM2  = SSMfar;                         % State Space matrix for interceptor #2
stateM2 = [0 0 0;                       % Initial StateMx
           0 0 0;                       % Initial StateMy
           0 0 0];                      % Initial StateMz
magGFM_2 = 0;                           % Magnitude of Guidance Force (N)
magncM_2 = 0;                           % Magnitude of the lat. acc.
xM_2 = [posM_2;vM_2];                   % State-Space matrix of Interceptor
%-------------------------------------------------------------------------

%Decoy for target- #1
hD_1 = 0;                               % Decoy Height (m)
magvD_1 = magvT_1;                      % Decoy Initial Velocity (m/s)
vD_1 = magvD_1 .* unitvD_1;             % Decoy Velocity Vector
groundTrackD_1 = posD_1;                % Decoy Ground Track Vector
distD_1 = 0;                            % Decoy Ground Distance (m)
oldGroundTrackD_1 = posT_1;             % Decoy previous ground track
xD1 = [posD_1; vD_1];                   % State-space Matrix of decoy-1

%Target-Decoy
distTD_1 = 0;                           % Target-Decoy Distance

%Decoy for target-2
hD_2 = 0;                               % Decoy Height (m)
magvD_2 = magvT_2;                      % Decoy Initial Velocity (m/s)
vD_2 = magvD_2 .* unitvD_2;             % Decoy Velocity Vector
groundTrackD_2 = posD_2;                % Decoy Ground Track Vector
distD_2 = 0;                            % Decoy Ground Distance (m)
oldGroundTrackD_2 = posT_2;             % Decoy previous ground track
xD2 = [posD_2; vD_2];                   % State-space Matrix of decoy-1

%Target-Decoy
distTD_2 = 0;                           % Target-Decoy Distance
%-------------------------------------------------------------------------

%Missile-#1--->Target-#1
VcMT_1 = 0;               % Missile-to-Target Sensed Closing Velocity (m/s)
VcMTTrue_1 = 0;           % Missile-to-Target True Closing Velocity (m/s)
distMT_1 = magnitude(posT_1 - posM_1);% Missile-to-Target Distance (m)
oldDistMTTrue_1 = 0;      % Previous True Missile-to-Target Distance (m)
oldLOSMT_1 = posT_1 - posM_1;% Previous Line of Sight (LOS)
oldDistMT_1 = magnitude(oldLOSMT_1);% Previous Target-Missile Distance

%Missile-#2--->Target-#2
VcMT_2 = 0;               % Missile-to-Target Sensed Closing Velocity (m/s)
VcMTTrue_2 = 0;           % Missile-to-Target True Closing Velocity (m/s)
distMT_2 = magnitude(posT_2 - posM_2);% Missile-to-Target Distance (m)
oldDistMTTrue_2 = 0;      % Previous True Missile-to-Target Distance (m)
oldLOSMT_2 = posT_2 - posM_2;% Previous Line of Sight (LOS)
oldDistMT_2 = magnitude(oldLOSMT_2);% Previous Target-Missile Distance
%-------------------------------------------------------------------------

%% Data Recording Arrays
tArray_1   = [];              % Simulation Time for plot Interception-#1
tArray_2   = [];              % Simulation Time for plot Interception-#2
%-------------------------------------------------------------------------

%Target-#1
posArrayT_1 = [];             % Target Position
sensedPosArrayT_1 = [];       % Sensed Target Position
trackingError_1 = [];         % Tracking position error
groundTrackArrayT_1 = [];     % Target Ground Track
distArrayT_1 = [];            % Target Ground Distance Array
hArrayT_1  = [];              % Target Height (m)
vArrayT_1  = [];              % Target Velocity (m/s)
stageArrayT_1 = [];           % Target Stage
massArrayT_1 = [];            % Target Total Mass
mnvrArrayT_1 = [];            % Target Maneuver

%Target-#2
```

```
posArrayT_2 = [];              % Target Position
sensedPosArrayT_2 = [];        % Sensed Target Position
trackingError_2 = [];          % Tracking position error
groundTrackArrayT_2 = [];      % Target Ground Track
distArrayT_2 = [];             % Target Ground Distance Array
hArrayT_2  = [];               % Target Height (m)
vArrayT_2  = [];               % Target Velocity (m/s)
stageArrayT_2 = [];            % Target Stage
massArrayT_2 = [];             % Target Total Mass
mnvrArrayT_2 = [];             % Target Maneuver
%--------------------------------------------------------------------------

%Missile-#1
posArrayM_1 = [];              % Missile Position
groundTrackArrayM_1 = [];      % Missile Ground Track
distArrayM_1 = [];             % Missile Ground Distance Array
hArrayM_1  = [];               % Missile Height (m)
vArrayM_1  = [];               % Missile Velocity (m/s)
stageArrayM_1 = [];            % Missile Stage
massArrayM_1 = [];             % Missile Total Mass
comLatAccArrayM_1 = [];        % Commanded Missile Lateral Acceleration (G)
achLatAccArrayM_1 = [];        % Achieved Missile Lateral Acceleration (G)
latDivArrayM_1 = [];           % Missile Lateral Divert (m/s)

%Missile-2
posArrayM_2 = [];              % Missile Position
groundTrackArrayM_2 = [];      % Missile Ground Track
distArrayM_2 = [];             % Missile Ground Distance Array
hArrayM_2  = [];               % Missile Height (m)
vArrayM_2  = [];               % Missile Velocity (m/s)
stageArrayM_2 = [];            % Missile Stage
massArrayM_2 = [];             % Missile Total Mass
comLatAccArrayM_2 = [];        % Commanded Missile Lateral Acceleration (G)
achLatAccArrayM_2 = [];        % Achieved Missile Lateral Acceleration (G)
latDivArrayM_2 = [];           % Missile Lateral Divert (m/s)
%--------------------------------------------------------------------------

%Decoy for Target-#1
posArrayD_1 = [];              % Decoy Position
groundTrackArrayD_1 = [];      % Decoy Ground Track
distArrayD_1 = [];             % Decoy Ground Distance Array
hArrayD_1  = [];               % Decoy Height (m)
vArrayD_1  = [];               % Decoy Velocity (m/s)

%Target-Decoy
distTDArray_1 = [];            % Target---> Decoy distance

%Decoy for Target-#2
posArrayD_2 = [];              % Decoy Position
groundTrackArrayD_2 = [];      % Decoy Ground Track
distArrayD_2 = [];             % Decoy Ground Distance Array
hArrayD_2  = [];               % Decoy Height (m)
vArrayD_2  = [];               % Decoy Velocity (m/s)

%Target-Decoy
distTDArray_2 = [];            % Target---> Decoy distance
%--------------------------------------------------------------------------

%Missile-1_Target-1
distArrayMT_1 = [];            % Missile-Target Distance
VcArrayMT_1 = [];              % Missile-Target Closing Velocity

%Missile-2_-Target-
distArrayMT_2 = [];            % Missile-Target Distance
VcArrayMT_2 = [];              % Missile-Target Closing Velocity

% Measurement and PDAF filter
targets = [];                  % Measurement empty tray
decoys = [];                   % Empty measurement tray
states = [];                   % Empty filter array
Ps = [];                       % Empty filter array
```

```
Fs = [];                        % Empty filter array

%% Simulation start here
while ((hT_1 >= SMALL) || (hM_1 >= SMALL)) && ((VcMT_1 >= 0)) ||...
      ((hT_2 >= SMALL) || (hM_2 >= SMALL)) && ((VcMT_2 >= 0))% Main loop

%% Target #1 motion
    if (t(1) >= lTimeT_1) &&((hT_1 >= SMALL) ||...
       (hM_1 >= SMALL)) && ((VcMT_1 >= 0)) % First intercepting
        % Initialize the Filter. By Assumption track is initializing by the
        % help of IR sensor
        if initializing(1)
            covi = [10;10;10;1;1;1];    % Initialization error
            x1ii = xT_1 + randn(size(covi)).*covi; % Initialize with SWAG
            states = [states, x1ii];
            Ps = [Ps, 10*diag(covi.^2)];% Initial state covariance
            initializing(1) = 0;        % Do not enter again
        end
        % Check for time step to change state space matrices
        if (distMT_1 <= timeStepSwitch) && timeStepFlag(1) && (t(1) > dt(1))
            dt(1) = nearTimeStep;  % Change Time steps for precision
            timeStepFlag(1) = 0;   % Reset Flag not to Enter Here Again
        end %if (distMT_1 <= timeStepSwitch) & timeStepFlag(1) & (t > dt)
        % When Target Launched (Target Computations)
        if (t(1) >= lTimeT_1) &&  (hT_1 >= SMALL)
            %Handle Target Stage Change Computations. Get new value
            if t(1) >= (stageChangeTimeT_1 + lTimeT_1)
                % Display Data on stage change
                disp(['Target #1 Stage-', num2str(stageT_1),...
                    ' Burnout: Speed = ', num2str(magnitude(vT_1)/1000),...
                    ' km/s, Altitude= ',  num2str(hT_1 / 1000), ' km.']);
                disp(['Target #1 and Interceptor #1 distance= ',...
                                        num2str(distMT_1/1000), ' km.']);
                disp(' ');
                stageT_1 = stageT_1 + 1;            % Increase Stage
                if stageT_1 >= size(dataMatrixT_1,2)% If No Next Stage
                    stageChangeTimeT_1 = BIG;
                else
                    % Set Next Stage Change Time
                    stageChangeTimeT_1 = ...
                        dataMatrixT_1(7, stageT_1 + 1);
                end
            end
            poserror_1 = magnitude(Hp*(xT_1-states(:,1)));   % Position error
            % Move the Target #1
            [F1, xT_1, hT_1, MT_1, posT_1, vT_1, aT_1, gT_1,...
            groundTrackT_1, oldGroundTrackT_1, distT_1, dataMatrixT_1] =...
                moveICBM(dt(1), xT_1, dataMatrixT_1, stageT_1,...
                          oldGroundTrackT_1, distT_1);
        end %(t >= lTimeT_1) &  (hT_1 >= SMALL) (When Target Launched)
        % Move Decoy
        if t(1) >= lTimeD_1
            [xD1, hD_1, posD_1, vD_1, groundTrackD_1, oldGroundTrackD_1,...
                distD_1, distTD_1] =...
            moveDecoys(dt(1), xD1, posT_1, distD_1, oldGroundTrackD_1);
            isDecoys(1) = 1;
        else    % If decoy does not released, then the same as ICBM
           xD1 = xT_1;                          % Decoys State
           posD_1 = Hp*xT_1;                    % Decoys position
           vD_1 = Hv*xT_1;                      % Decoys velocity
           hD_1 = hT_1;                         % decoys altitude
           groundTrackD_1 = groundTrackT_1;     % Decoys Ground track
           oldGroundTrackD_1 = groundTrackT_1;  % Decoys Old Ground Track
           distD_1 = distT_1;                   % Decoys range
           distTD_1 = 0;                        % Decoy --> UCBM Distance
        end

        % Record Data
        tArray_1 = [tArray_1 t(1)];             % Simulation time
        % Target
        posArrayT_1 = [posArrayT_1 posT_1];     % Target position
```

```
        sensedPosArrayT_1 = ...
            [sensedPosArrayT_1 sensedPosT_1];   % Sensed target position
        trackingError_1 =...
            [trackingError_1 poserror_1];       % Position error
        hArrayT_1 = [hArrayT_1 hT_1];           % Target height
        groundTrackArrayT_1 =...
            [groundTrackArrayT_1 groundTrackT_1];% Target ground track
        distArrayT_1 = [distArrayT_1 distT_1];   % Target downrange
        vArrayT_1 = [vArrayT_1 magnitude(vT_1)]; % Target velocity
        stageArrayT_1 = [stageArrayT_1 stageT_1];% Target stage
        massArrayT_1 = [massArrayT_1 MT_1];      % Target mass
        mnvrArrayT_1 = [mnvrArrayT_1 mnvrT_1];   % Target maneuver
        %Missile
        posArrayM_1 = [posArrayM_1 posM_1];      % Missile position
        hArrayM_1 = [hArrayM_1 hM_1];            % Missile height
        groundTrackArrayM_1 =...
            [groundTrackArrayM_1 groundTrackM_1];% Missile ground track
        distArrayM_1 = [distArrayM_1 distM_1];   % Missile downrange
        vArrayM_1 = [vArrayM_1 magnitude(vM_1)]; % Missile velocity
        stageArrayM_1 = [stageArrayM_1 stageM_1];% Missile stage
        massArrayM_1 = [massArrayM_1 MM_1];      % Missile Mass
        comLatAccArrayM_1 =...
            [comLatAccArrayM_1 comLatAccM_1];    % Commanded lat. acc.
        achLatAccArrayM_1 =...
            [achLatAccArrayM_1 achLatAccM_1];    % Achieved lat. acc.
        latDivArrayM_1 = [latDivArrayM_1 latDivM_1];% .Lateral divert
        %Decoy
        posArrayD_1 = [posArrayD_1 posD_1];      % Decoy position
        hArrayD_1 = [hArrayD_1 hD_1];            % Decoy height
        groundTrackArrayD_1 =...
            [groundTrackArrayD_1 groundTrackD_1];% Decoy ground track
        distArrayD_1 = [distArrayD_1 distD_1];   % Decoy downrange
        vArrayD_1 = [vArrayD_1 magnitude(vD_1)]; % Decoy velocity
        %Target-Decoy
        distTDArray_1 = [distTDArray_1 distTD_1];% Target-Decoy distance
        %Missile-Target
        distArrayMT_1 = [distArrayMT_1 distMT_1];% Missile-target distance
        VcArrayMT_1 = [VcArrayMT_1 VcMT_1];      % Closing velocity
    end
%% Target #2 motion
    if (t(2) >= lTimeT_2) && ((hT_2 >= SMALL) ||...
       (hM_2 >= SMALL)) && ((VcMT_2 >= 0))  % First intercepting
        % Initialize the Filter. By Assumption track is initializing by the
        % help of IR sensor
        if initializing(2)
            covi = [10;10;10;1;1;1];         % Initialization error
            x2ii = xT_2 + randn(size(covi)).*covi; % Star with SWAG
            states = [states, x2ii];
            Ps = [Ps, 10*diag(covi.^2)] ;    % Initialize with big inaccuracy
            initializing(2) = 0;             % Do not initialize again
        end
        % Check for time step to change state space matrices
        if (distMT_2 <= timeStepSwitch) && timeStepFlag(2) && (t(2) > dt(2))
            dt(2) = nearTimeStep;  % Change Time steps for precision
            timeStepFlag(2) = 0;   % Reset Flag not to Enter Here Again
        end %if (distMT_2 <= timeStepSwitch) & timeStepFlag(2) & (t > dt)
        % When Target Launched (Target Computations)
        if (t(2) >= lTimeT_2) &&  (hT_2 >= SMALL)
            %Handle Target Stage Change Computations. Get new value
            if t(2) >= (stageChangeTimeT_2 + lTimeT_2)
                % Display Data on stage change
                disp(['Target #2 Stage-', num2str(stageT_2),...
                    ' Burnout: Speed = ', num2str(magnitude(vT_2)/1000),...
                    ' km/s, Altitude= ',  num2str(hT_2 / 1000) ' km.']);
                disp(['Target #2 and Interceptor #2 distance = '...
                                        num2str(distMT_2/1000), ' km.']);
                disp(' ');
                stageT_2 = stageT_2 + 1;                  % Increase Stage
                if stageT_2 >= size(dataMatrixT_2,2)      % If No Next Stage
                    stageChangeTimeT_2 = BIG;
                else
```

```
                stageChangeTimeT_2 = ...
                      dataMatrixT_2(7, stageT_2 + 1); % Set Next
            end
        end
        poserror_2 = magnitude(Hp*(xT_2-states(:,2)));   % Position error
        % Move the ICBM
        [F2, xT_2, hT_2, MT_2, posT_2, vT_2, aT_2, gT_2,...
        groundTrackT_2, oldGroundTrackT_2, distT_2, dataMatrixT_2] =...
            moveICBM(dt(2), xT_2, dataMatrixT_2, stageT_2, ...
                     oldGroundTrackT_2, distT_2);
    end %(t >= lTimeT_1) &  (hT_1 >= SMALL) (When Target Launched)
    if t(2) >= lTimeD_2
        [xD2, hD_2, posD_2, vD_2, groundTrackD_2, oldGroundTrackD_2,...
            distD_2, distTD_2] =...
            moveDecoys(dt(2), xD2, posT_2, distD_2, oldGroundTrackD_2);
        isDecoys(2) = 1;
    else    % If decoy does not released, then everything equals to ICBM
       xD2 = xT_2;                          % Decoys State
       posD_2 = Hp*xT_2;                    % Decoys position
       vD_2 = Hv*xT_2;                      % Decoys velocity
       hD_2 = hT_2;                         % decoys altitude
       groundTrackD_2 = groundTrackT_2;     % Decoys Ground track
       oldGroundTrackD_2 = groundTrackT_2;  % Decoys Old Ground Track
       distD_2 = distT_2;                   % Decoys range
       distTD_2 = 0;                        % Decoy --> ICBM Distance
    end

    %Record Data
    tArray_2 = [tArray_2 t(2)];             % Simulation time
    %Target
    posArrayT_2 = [posArrayT_2 posT_2];     % Target position
    sensedPosArrayT_2 =...
        [sensedPosArrayT_2 sensedPosT_2];   % Sensed target position
    trackingError_2 =...
        [trackingError_2 poserror_2];       % Position error
    hArrayT_2 = [hArrayT_2 hT_2];           % Target height
    groundTrackArrayT_2 =...
        [groundTrackArrayT_2 groundTrackT_2];% Target ground track
    distArrayT_2 = [distArrayT_2 distT_2];   % Target downrange
    vArrayT_2 = [vArrayT_2 magnitude(vT_2)]; % Target velocity
    stageArrayT_2 = [stageArrayT_2 stageT_2];% Target stage
    massArrayT_2 = [massArrayT_2 MT_2];      % Target mass
    mnvrArrayT_2 = [mnvrArrayT_2 mnvrT_2];   % Target maneuver
    %Missile
    posArrayM_2 = [posArrayM_2 posM_2];      % Missile position
    hArrayM_2 = [hArrayM_2 hM_2];            % Missile height
    groundTrackArrayM_2 =...
        [groundTrackArrayM_2 groundTrackM_2];% Missile ground track
    distArrayM_2 = [distArrayM_2 distM_2];   % Missile downrange
    vArrayM_2 = [vArrayM_2 magnitude(vM_2)]; % Missile velocity
    stageArrayM_2 = [stageArrayM_2 stageM_2];% Missile stage
    massArrayM_2 = [massArrayM_2 MM_2];      % Missile Mass
    comLatAccArrayM_2 =...
        [comLatAccArrayM_2 comLatAccM_2];    % Commanded lat. acc.
    achLatAccArrayM_2 =...
        [achLatAccArrayM_2 achLatAccM_2];    % Achieved lat. acc.
    latDivArrayM_2 = [latDivArrayM_2 latDivM_2];% Lateral divert
    %Decoy
    posArrayD_2 = [posArrayD_2 posD_2];      % Decoy position
    hArrayD_2 = [hArrayD_2 hD_2];            % Decoy height
    groundTrackArrayD_2 =...
        [groundTrackArrayD_2 groundTrackD_2];% Decoy ground track
    distArrayD_2 = [distArrayD_2 distD_2];   % Decoy downrange
    vArrayD_2 = [vArrayD_2 magnitude(vD_2)]; % Decoy velocity
    %Target-Decoy
    distTDArray_2 = [distTDArray_2 distTD_2];% Target-Decoy distance
    %Missile-Target
    distArrayMT_2 = [distArrayMT_2 distMT_2];% Missile-target distance
    VcArrayMT_2 = [VcArrayMT_2 VcMT_2];      % Closing velocity
end
```

```
%% Measurement and Fusion
    if ~initializing(1)
        targets = [targets, xT_1];
        decoys = [decoys, xD1];
    end
    if ~initializing(2)
        targets = [targets, xT_2];
        decoys = [decoys, xD2];
    end

    % Only take measurement if any track initialized
    if ~initializing(1) || ~initializing(2)
        %RF sensor look angle and distance to target and sensed position
        % calculation
        [measurements1, covariance1] = Scope(isDecoys, isReduced,...
            isJamming, targets, decoys, posRF1);
        [measurements2, covariance2] = Scope(isDecoys, isReduced,...
            isJamming, targets, decoys, posRF2);
        [measurements3, covariance3] = Scope(isDecoys, isReduced,...
            isJamming, targets, decoys, posRF3);
        targets = [];                       % Empty it again for next step
        decoys = [];                        % Empty it for the next step
    end
    %% Prepare data for the fusion box
    if ~initializing(1)
        Fs = [Fs, F1];
    end
    if ~initializing(2)
        Fs = [Fs, F2];
    end
    % Only take measurement if any track initialized
    if ~initializing(1) || ~initializing(2)
        measurements = [measurements1;
                        measurements2;
                        measurements3];     % Create the measurements stack
        covariances = [covariance1;
                       covariance2;
                       covariance3];        % Covariance stack
        % Correlate, Associate, Filter and Fuse the measurements
        [states, Ps] = Mash(isJamming, states, Ps, Fs,...
                            measurements, covariances);
        Fs = [];                            % Set empty so that load new ones
    end

%% Interceptor #1 Motion
    if (t(1) >= lTimeM_1) && ((hT_1 >= SMALL) ||...
       (hM_1 >= SMALL)) &&  (VcMT_1 >= 0)%When Missile #1 Launched
        if launch_flag(1)
            % Indicate Interceptor Launched
            disp('Interception #1');
            disp('First Interceptor launched!');
            disp(' ')
            launch_flag(1) = 0;
        end

        %Handle Target Stage Change Computations
        if t(1) >= (stageChangeTimeM_1 + lTimeM_1)
            % Display Data on stage change
            disp(['Interceptor #1 Stage-' num2str(stageM_1)...
                  ' Burnout: Speed = ' num2str(magnitude(vM_1)/1000)...
                  ' km/s, Altitude= '  num2str(hM_1 / 1000) ' km.']);
            disp(['Target #1 and Interceptor #1  distance= ',...
                                        num2str(distMT_1/1000), ' km.']);
            disp(' ');
            stageM_1 = stageM_1 + 1;                    % Increase Stage
            if stageM_1 >= size(dataMatrixM_1,2)        % If No Next Stage
                stageChangeTimeM_1 = BIG;
            else
                stageChangeTimeM_1 = ...
                    dataMatrixM_1(7, stageM_1 + 1);     % Set Next
            end
```

```
        end

        % Launch Kill vehicle
        if (distMT_1 <= timeStepSwitch)  && (kill_launch(1))
            stageM_1 = size(dataMatrixM_1, 2);  % This the final stage
            dt(1) = nearTimeStep;
            disp(' ')
            % Display Data on stage change
            disp(['KILL VEHICLE IS LAUNCHED FOR INTERCEPTION #1 TIME ',...
                                        num2str(t(1)), ' seconds']);
            disp(['Kill Vehicle #1 Stage-' num2str(stageM_1)...
                  ' Burnout: Speed = ' num2str(magnitude(vM_1)/1000)...
                  ' km/s, Altitude= '  num2str(hM_1 / 1000) ' km.']);
            disp(['Target and Kill Vehicle Distance = ',...
                                        num2str(distMT_1/1000), ' km.']);
            disp(' ');
            stageChangeTimeM_1 = BIG;        % never change the stage again
            dMdtM_1 = 5;                     % JUST FOR GUIDANCE
            txDelay(1) = 0; % Using onboard sensor  no transfer delay
            maxG(1) = 15;   % The kill vehicle more capable to maneuver
            navCoefM(1) = 5;% The kill vehicle more capable to maneuver
            kill_launch(1) = 0;              % Never enter here again
            updateTime(1) = nearTimeStep*1;  % Sensor Update Interval (s)
            SSM1  = SSMnear;                 % Switch the Near time step
            timeFlags  = [t;
                          dt;
                          txFlag;
                          txCounter;
                          txDelay];          % Update time flag info
        end
        % Move the Interceptor
          [xM_1, dataMatrixM_1, posM_1, vM_1, unitvM_1, gM_1, aM_1,...
              hM_1, MM_1, distM_1, groundTrackM_1, oldGroundTrackM_1] =...
         moveInterceptor(dt(1), xM_1, dataMatrixM_1, stageM_1,...
                         GFM_1, oldGroundTrackM_1, distM_1);
    end %(t >= lTimeM_1) & (hM_1 >= SMALL) (When Missile Launched)
    % Exploit proportional Guidance
    if ~initializing(1)
        sensedPosT_1 = Hp*states(:,1);         % First in the stack
        if ~(kill_launch(1))
            % The onboard sensors are more sensitive that the max tolerable std
            % dev is 0.5 m which can be neglected here. so the sensed position
            % will be target position if the kill vehicle launched
            %               sensedPosT_1 = posT_1 ;
            sensedPosT_1 = posT_1 + randn(3,1)*0.5;
        end
        % Calculate the Guidance force to maneuver interceptor.
        [distMT_1, VcMTTrue_1, oldDistMTTrue_1, VcMT_1, oldDistMT_1,...
            oldLOSMT_1, mnvrT_1, nlM_1, GFM_1, magGFM_1, magncM_1, ...
               comLatAccM_1, achLatAccM_1, latDivM_1] =...
        guidance(1, posT_1, posM_1, sensedPosT_1, vT_1, VcMT_1, ...
                lTimeM_1, oldDistMTTrue_1, oldDistMT_1, oldLOSMT_1,aT_1,...
                 gT_1, nlM_1, gM_1, MM_1, unitvM_1, latDivM_1, magncM_1);
    end

%% Interceptor #2 motion
    if (t(2) >= lTimeM_2) && ((hT_2 >= SMALL) ||...
       (hM_2 >= SMALL)) &&  (VcMT_2 >= 0)  %When Missile #2 Launched
        if launch_flag(2)
            % Indicate Interceptor Launched
            disp('Interception #2');
            disp('Second Interception launched!');
            disp(' ')
            launch_flag(2) = 0;
        end

        %Handle Target Stage Change Computations
        if t(2) >= (stageChangeTimeM_2 + lTimeM_2)
            % Display Data on stage change
            disp(['Interceptor #2 Stage-' num2str(stageM_2)...
                  ' Burnout: Speed = ' num2str(magnitude(vM_2)/1000)...
```

```
                ' km/s, Altitude= '  num2str(hM_2 / 1000) ' km.']);
            disp(['Target #2 and Interceptor #2 distance = '...
                                    num2str(distMT_2/1000), ' km.']);
            disp(' ');
            stageM_2 = stageM_2 + 1;                    % Increase Stage
            if stageM_2 >= size(dataMatrixM_2,2)        % If No Next Stage
                stageChangeTimeM_2 = BIG;
            else
                stageChangeTimeM_2 = ...
                    dataMatrixM_2(7, stageM_2 + 1);
            end
        end

        % Launch Kill vehicle
        if (distMT_2 <= timeStepSwitch) && (kill_launch(2))
            stageM_2 = size(dataMatrixM_2, 2);  % This the final stage
            dt(2) = nearTimeStep;
            disp(' ')
            % Display Data on stage change
            disp(['KILL VEHICLE IS LAUNCHED FOR INTERCEPTION #2 TIME ',...
                                num2str(t(2)), ' seconds']);
            disp(['Kill Vehicle #2 Stage-' num2str(stageM_2)...
                  ' Burnout: Speed = ' num2str(magnitude(vM_2)/1000)...
                  ' km/s, Altitude= '  num2str(hM_2 / 1000) ' km.']);
            disp(['Target #2 and Kill Vehicle #2 distance = '...
                                    num2str(distMT_2/1000), ' km.']);
            disp(' ');
            stageChangeTimeM_2 = BIG;% never change the stage again
            dMdtM_2 =  5;            % JUST FOR GUIDANCE
            txDelay(1) = 0; % Using onboard sensor no transfer delay
            maxG(2) = 15;            % The kill vehicle more capable to maneuver
            navCoefM(2) = 5;         % The kill vehicle more capable to maneuver
            kill_launch(2) = 0;      % Never enter here again
            updateTime(2) = nearTimeStep*1;% Sensor Update Interval (s)
            SSM2  = SSMnear;         % Switch Near time step
            timeFlags  = [t;
                          dt;
                          txFlag;
                          txCounter;
                          txDelay];   % Update time flag info
        end
        % Move the Interceptor
        [xM_2, dataMatrixM_2, posM_2, vM_2, unitvM_2, gM_2, aM_2, hM_2,...
            MM_2, distM_2, groundTrackM_2, oldGroundTrackM_2] =...
         moveInterceptor(dt(2), xM_2, dataMatrixM_2, stageM_2,...
                         GFM_2, oldGroundTrackM_2, distM_2);
    end %(When Missile Launched)
    % Exploit proportional Guidance
    if ~initializing(2)
        sensedPosT_2 = Hp*states(:,2);         % First in the stack
        if ~(kill_launch(2))
            % The onboard sensors are more sensitive that the max tolerable std
            % dev is 0.5 m which can be neglected here. so the sensed position
            % will be target position if the kill vehicle launched
            %               sensedPosT_2 = posT_2;
            sensedPosT_2 = posT_2 + randn(3,1)*0.5;
        end
        % Calculate the Guidance force to maneuver interceptor.
        [distMT_2, VcMTTrue_2, oldDistMTTrue_2, VcMT_2, oldDistMT_2,...
            oldLOSMT_2, mnvrT_2, nlM_2, GFM_2, magGFM_2, magncM_2,...
                comLatAccM_2, achLatAccM_2, latDivM_2] =...
        guidance(2, posT_2, posM_2, sensedPosT_2, vT_2, VcMT_2,...
               lTimeM_2, oldDistMTTrue_2, oldDistMT_2, oldLOSMT_2, aT_2,...
               gT_2, nlM_2, gM_2, MM_2, unitvM_2, latDivM_2, magncM_2);
    end
%% Increase the time
    t = timeFlags(1,:);                     % Take the times to increase
    t = t + timeFlags(2,:);                 % Increase Time
    timeFlags(1,:) = t;                     % Set the time Flags
    txCounter = timeFlags(4,:);             % Take the Counter to increase
    txCounter = txCounter + timeFlags(2,:);   % Increase Counter
```

```
    timeFlags(4,:) = txCounter;              % Set the time Flag
    if t >= 3600                             % Exit After 1 Hour Anyway
        break;
    end
end % (Main loop)

%% Erase Data After Miss
passIndex_1 = find(distArrayMT_1 == min(distArrayMT_1))-1;
%Target
tArray_1 = tArray_1(:, 1:passIndex_1);
posArrayT_1 = posArrayT_1(:, 1:passIndex_1);
sensedPosArrayT_1 = sensedPosArrayT_1(:, 1:passIndex_1);
trackingError_1 = trackingError_1(:,1:passIndex_1);
hArrayT_1 = hArrayT_1(:, 1:passIndex_1);
groundTrackArrayT_1 = groundTrackArrayT_1(:, 1:passIndex_1);
distArrayT_1 = distArrayT_1(:, 1:passIndex_1);
vArrayT_1 = vArrayT_1(:, 1:passIndex_1);
stageArrayT_1 = stageArrayT_1(:, 1:passIndex_1);
massArrayT_1 = massArrayT_1(:, 1:passIndex_1);
mnvrArrayT_1 = mnvrArrayT_1(:, 1:passIndex_1);
%Missile
posArrayM_1 = posArrayM_1(:, 1:passIndex_1);
hArrayM_1 = hArrayM_1(:, 1:passIndex_1);
groundTrackArrayM_1 = groundTrackArrayM_1(:, 1:passIndex_1);
distArrayM_1 = distArrayM_1(:, 1:passIndex_1);
vArrayM_1 = vArrayM_1(:, 1:passIndex_1);
stageArrayM_1 = stageArrayM_1(:, 1:passIndex_1);
massArrayM_1 = massArrayM_1(:, 1:passIndex_1);
comLatAccArrayM_1 = comLatAccArrayM_1(:, 1:passIndex_1);
achLatAccArrayM_1 = achLatAccArrayM_1(:, 1:passIndex_1);
latDivArrayM_1 = latDivArrayM_1(:, 1:passIndex_1);
%Decoy
posArrayD_1 = posArrayD_1(:, 1:passIndex_1);
hArrayD_1 = hArrayD_1(:, 1:passIndex_1);
groundTrackArrayD_1 = groundTrackArrayD_1(:, 1:passIndex_1);
distArrayD_1 = distArrayD_1(:, 1:passIndex_1);
vArrayD_1 = vArrayD_1(:, 1:passIndex_1);
%Target-Decoy
distTDArray_1 = distTDArray_1(:, 1:passIndex_1);
%Missile-Target
distArrayMT_1 = distArrayMT_1(:, 1:passIndex_1);
VcArrayMT_1 = VcArrayMT_1(:, 1:passIndex_1);
%--------------------------------------------------------------------------

%Erase Data After Miss
passIndex_2 = find(distArrayMT_2 == min(distArrayMT_2))-1;
%Target
tArray_2 = tArray_2(:, 1:passIndex_2);
posArrayT_2 = posArrayT_2(:, 1:passIndex_2);
sensedPosArrayT_2 = sensedPosArrayT_2(:, 1:passIndex_2);
trackingError_2 = trackingError_2(:,1:passIndex_2);
hArrayT_2 = hArrayT_2(:, 1:passIndex_2);
groundTrackArrayT_2 = groundTrackArrayT_2(:, 1:passIndex_2);
distArrayT_2 = distArrayT_2(:, 1:passIndex_2);
vArrayT_2 = vArrayT_2(:, 1:passIndex_2);
stageArrayT_2 = stageArrayT_2(:, 1:passIndex_2);
massArrayT_2 = massArrayT_2(:, 1:passIndex_2);
mnvrArrayT_2 = mnvrArrayT_2(:, 1:passIndex_2);
%Missile
posArrayM_2 = posArrayM_2(:, 1:passIndex_2);
hArrayM_2 = hArrayM_2(:, 1:passIndex_2);
groundTrackArrayM_2 = groundTrackArrayM_2(:, 1:passIndex_2);
distArrayM_2 = distArrayM_2(:, 1:passIndex_2);
vArrayM_2 = vArrayM_2(:, 1:passIndex_2);
stageArrayM_2 = stageArrayM_2(:, 1:passIndex_2);
massArrayM_2 = massArrayM_2(:, 1:passIndex_2);
comLatAccArrayM_2 = comLatAccArrayM_2(:, 1:passIndex_2);
achLatAccArrayM_2 = achLatAccArrayM_2(:, 1:passIndex_2);
latDivArrayM_2 = latDivArrayM_2(:, 1:passIndex_2);
%Decoy
posArrayD_2 = posArrayD_2(:, 1:passIndex_2);
```

```
hArrayD_2 = hArrayD_2(:, 1:passIndex_2);
groundTrackArrayD_2 = groundTrackArrayD_2(:, 1:passIndex_2);
distArrayD_2 = distArrayD_2(:, 1:passIndex_2);
vArrayD_2 = vArrayD_2(:, 1:passIndex_2);
%Target-Decoy
distTDArray_2 = distTDArray_2(:, 1:passIndex_2);
%Missile-Target
distArrayMT_2 = distArrayMT_2(:, 1:passIndex_2);
VcArrayMT_2 = VcArrayMT_2(:, 1:passIndex_2);
% -------------------------------------------------------------------------

%% Plot What you have
%Define Earth
[xE, yE, zE] = sphere(36);                  % Generate Unit Sphere
xE = xE .* Re;                              % Expand X-axis
yE = yE .* Re;                              % Expand Y-axis
zE = zE .* Re;                              % Expand Z-axis

%Visualize Earth & The Coordinate Frame
figure;
axis equal;
axis([-7e6 7e6 -7e6 7e6 -7e6 7e6]);        % Set Axes
view(280,30);                               % Set Suitable View
grid on;
hold on;
surf(xE, yE, zE);                           % Plot Earth

%3D Target Trajectory and Ground Track
title('Trajectories & Ground tracks')
xlabel('x(m)');
ylabel('y(m)');
zlabel('z(m)');

% Plot Target Trajectory
plot3(posArrayT_1(1,:), posArrayT_1(2,:), posArrayT_1(3,:), 'y-');

%Plot Target_2 Trajectory
plot3(posArrayT_2(1,:), posArrayT_2(2,:), posArrayT_2(3,:), 'y-');

% Plot Missile Trajectory
plot3(posArrayM_1(1,:), posArrayM_1(2,:), posArrayM_1(3,:), 'b--');

% Plot Missile Trajectory
plot3(posArrayM_2(1,:), posArrayM_2(2,:), posArrayM_2(3,:), 'b--');

% Plot Decoy Trajectory
plot3(posArrayD_1(1,:), posArrayD_1(2,:), posArrayD_1(3,:), 'g-.');

% Plot Decoy Trajectory
plot3(posArrayD_2(1,:), posArrayD_2(2,:), posArrayD_2(3,:), 'g-.');

% Plot Target Ground track
plot3(groundTrackArrayT_1(1,:), groundTrackArrayT_1(2,:),...
      groundTrackArrayT_1(3,:), 'k:');

% Plot Target Ground track
plot3(groundTrackArrayT_2(1,:), groundTrackArrayT_2(2,:),...
      groundTrackArrayT_2(3,:), 'k:');

% Plot Missile Ground track
plot3(groundTrackArrayM_1(1,:), groundTrackArrayM_1(2,:),...
      groundTrackArrayM_1(3,:), 'k:');

% Plot Missile Ground track
plot3(groundTrackArrayM_2(1,:), groundTrackArrayM_2(2,:),...
      groundTrackArrayM_2(3,:), 'k:');

% Plot Decoy Ground track
plot3(groundTrackArrayD_1(1,:), groundTrackArrayD_1(2,:),...
      groundTrackArrayD_1(3,:), 'k:');
```

```
% Plot Decoy Ground track
plot3(groundTrackArrayD_2(1,:), groundTrackArrayD_2(2,:),...
      groundTrackArrayD_2(3,:), 'k:');

plot3(pos0T_1(1), pos0T_1(2), pos0T_1(3), 'bo')
plot3(pos0M_1(1), pos0M_1(2), pos0M_1(3), 'go')
plot3(pos0T_2(1), pos0T_2(2), pos0T_2(3), 'bo')
plot3(pos0M_2(1), pos0M_2(2), pos0M_2(3), 'go')
plot3(posRF1(1), posRF1(2), posRF1(3), 'co');
plot3(posRF2(1), posRF2(2), posRF2(3), 'mo');
plot3(posRF3(1), posRF3(2), posRF3(3), 'yo');

% Plot Distance vs. Height
figure;
hold on;
plot((distArrayT_1 ./ 1000), (hArrayT_1 ./ 1000),'r-');
plot((distArrayM_1 ./ 1000), (hArrayM_1 ./ 1000),'b--');
plot((distArrayD_1 ./ 1000), (hArrayD_1 ./ 1000),'g-.');
title('Ground Distance vs. Height for Interception #1');
xlabel('Ground Distance (km)');
ylabel('Height (km)');
legend('Target #1','Interceptor #1', 'Decoy #1', 0);

% Plot Distance vs. Height
figure;
hold on;
plot((distArrayT_2 ./ 1000), (hArrayT_2 ./ 1000),'r-');
plot((distArrayM_2 ./ 1000), (hArrayM_2 ./ 1000),'b--');
plot((distArrayD_2 ./ 1000), (hArrayD_2 ./ 1000),'g-.');
title('Ground Distance vs. Height for Interception #2');
xlabel('Ground Distance (km)');
ylabel('Height (km)');
legend('Target #2','Interceptor #2', 'Decoy #2', 0);

% Plot Time vs. Height
figure
hold on;
plot((tArray_1 ./ 60), (hArrayT_1 ./ 1000),'r-');
plot((tArray_1 ./ 60), (hArrayM_1 ./ 1000),'b--');
plot((tArray_1 ./ 60), (hArrayD_1 ./ 1000),'g-.');
title('Time vs. Height for Interception #1');
xlabel('Flight Time (min)');
ylabel('Height (km)');
legend('Target #1','Interceptor #1', 'Decoy #1', 0);

% Plot Time versus Height
figure
hold on;
plot((tArray_2 ./ 60), (hArrayT_2 ./ 1000),'r-');
plot((tArray_2 ./ 60), (hArrayM_2 ./ 1000),'b--');
plot((tArray_2 ./ 60), (hArrayD_2 ./ 1000),'g-.');
title('Time vs. Height for interception #2');
xlabel('Flight Time (min)');
ylabel('Height (km)');
legend('Target #2','Interceptor #2', 'Decoy #2', 0);

% Plot Speed vs. Time
figure;
hold on;
plot((tArray_1 ./ 60), (vArrayT_1 ./ 1000),'r-');
plot((tArray_1 ./ 60), (vArrayM_1 ./ 1000),'b--');
plot((tArray_1 ./ 60), (vArrayD_1 ./ 1000),'g-.');
title('Velocity vs. Flight Time for interception #1');
xlabel('Flight Time (min)');
ylabel('Velocity (km/s)');
legend('Target #1','Interceptor #1','Decoy #1', 0);

% Plot Speed vs. Time
figure;
hold on;
plot((tArray_2 ./ 60), (vArrayT_2 ./ 1000),'r-');
```

```
plot((tArray_2 ./ 60), (vArrayM_2 ./ 1000),'b--');
plot((tArray_2 ./ 60), (vArrayD_2 ./ 1000),'g-.');
title('Velocity vs. Flight Time for interception #2');
xlabel('Flight Time (min)');
ylabel('Velocity (km/s)');
legend('Target #2','Interceptor #2','Decoy #2', 0);

% Plot Stage vs. Time
figure;
hold on;
plot((tArray_1 ./ 60), stageArrayT_1,'r');
plot((tArray_1 ./ 60), stageArrayM_1,'b');
title('Stage vs. Flight Time for Interception #1');
xlabel('Flight Time (min)');
ylabel('Stage');
legend('Target #1','Interceptor #1', 0);

% Plot Stage vs. Time
figure;
hold on;
plot((tArray_2 ./ 60), stageArrayT_2,'r');
plot((tArray_2 ./ 60), stageArrayM_2,'b');
title('Stage vs. Flight Time for interception #2');
xlabel('Flight Time (min)');
ylabel('Stage');
legend('Target #2','Interceptor #2', 0);

% Plot Total Mass vs. Time
figure;
hold on;
plot((tArray_1 ./ 60), massArrayT_1 ./ 1000,'r');
plot((tArray_1 ./ 60), massArrayM_1 ./ 1000,'b');
title('Total Mass vs. Flight Time for Interception #1');
xlabel('Flight Time (min)');
ylabel('Mass (Tons)');
legend('Target #1','Interceptor #1', 0);

% Plot Total Mass vs. Time
figure;
hold on;
plot((tArray_2 ./ 60), massArrayT_2 ./ 1000,'r');
plot((tArray_2 ./ 60), massArrayM_2 ./ 1000,'b');
title('Total Mass vs. Flight Time for interception #2');
xlabel('Flight Time (min)');
ylabel('Mass (Tons)');
legend('Target #2','Interceptor #2', 0);

% Plot Missile-Target Distance
figure;
plot((tArray_1 ./ 60), distArrayMT_1 ./ 1000,'b');
title('Missile-Target Distance for Interception #1');
xlabel('Time (min)');
ylabel('Distance (km)');

% Plot Missile-Target Distance
figure;
plot((tArray_2 ./ 60), distArrayMT_2 ./ 1000,'b');
title('Missile-Target Distance for Interception #2');
xlabel('Time (min)');
ylabel('Distance (km)');

% Plot Missile-Target Closing Velocity
figure;
plot((tArray_1 ./ 60), VcArrayMT_1 ./ 1000,'b');
axis([0 3 0 14]);
title('Missile-Target Closing Velocity for Interception #1');
xlabel('Time (min)');
ylabel('Vc (km/s)');

% Plot Missile-Target Closing Velocity
figure;
```

```
plot((tArray_2 ./ 60), VcArrayMT_2 ./ 1000,'b');
axis([0 3  0 14]);
title('Missile-Target Closing Velocity for interception #2');
xlabel('Time (min)');
ylabel('Vc (km/s)');

%Plot Missile Lateral Acceleration
figure;
hold on
plot((tArray_1 ./ 60), comLatAccArrayM_1 ,'b');
plot((tArray_1 ./ 60), achLatAccArrayM_1 ,'r');
title('Missile Lateral Acceleration for Interception #1');
xlabel('Time (min)');
ylabel('Lateral Acceleration (g)');
axis([0 3 0 15]);
legend('Commanded', 'Achieved', 0);

% Plot Missile Lateral Acceleration
figure;
hold on
plot((tArray_2 ./ 60), comLatAccArrayM_2 ,'b');
plot((tArray_2 ./ 60), achLatAccArrayM_2 ,'r');
title('Missile Lateral Acceleration for Interception #2');
xlabel('Time (min)');
ylabel('Lateral Acceleration (g)');
axis([0 3 0 15]);
legend('Commanded', 'Achieved', 0);

% Plot Missile Lateral Divert
figure;
plot((tArray_1 ./ 60), latDivArrayM_1 ,'b');
title('Missile Lateral Divert for Interception #1');
xlabel('Time (min)');
ylabel('Lateral Divert (m/s)');

% Plot Missile Lateral Divert
figure;
plot((tArray_2 ./ 60), latDivArrayM_2 ,'b');
title('Missile Lateral Divert for Interception #1');
xlabel('Time (min)');
ylabel('Lateral Divert for interception 2 (m/s)');

% Plot Target Maneuver
figure;
plot((tArray_1 ./ 60), mnvrArrayT_1 ,'b');
title('Target #1 Maneuver ');
xlabel('Time (min)');
ylabel('Maneuver (g)');

% Plot Target Maneuver
figure;
plot((tArray_2 ./ 60), mnvrArrayT_2 ,'b');
title('Target #2 Maneuver');
xlabel('Time (min)');
ylabel('Maneuver (g)');

% Position Error
figure;
plot((tArray_1 ./ 60), trackingError_1 ,'b-');
title('Position Error for Target #1');
xlabel('Time (min)');
ylabel('Magnitude of Position Error (m)');
axis([0 3 0 max(trackingError_1)]);

%Position Error
figure;
plot((tArray_2 ./ 60), trackingError_2 ,'b-');
title('Position Error for Target #2');
xlabel('Time (min)');
ylabel('Magnitude of Position Error (m)');
axis([0 3 0 max(trackingError_2)]);
```

```
% Target-Decoy Distance
figure;
plot((tArray_1 ./ 60), (distTDArray_1./1000) ,'b-');
title('Target #1-Decoy #1 Separation');
xlabel('Time (min)');
ylabel('Target-Decoy Distance (km)');

% Target-Decoy Distance
figure;
plot((tArray_2 ./ 60), (distTDArray_2./1000) ,'b-');
title('Target #2-Decoy #2 Separation');
xlabel('Time (min)');
ylabel('Target-Decoy Distance (km)');

%Display Final Intercept Data
disp('INTERCEPTION #1');
disp (['Target Range =' num2str(distT_1 / 1000) ' km.'])
disp (['Missile Range =' num2str(distM_1 / 1000) ' km.'])
disp (' ' );
disp (['Intercept Time =' num2str(max(tArray_1)/60) ' minutes.'])
disp (['Miss Distance =' num2str(min(distArrayMT_1)) ' m.'])
disp (['Lateral Divert =' num2str(max(latDivArrayM_1)) ' m/s.'])
disp (' ');

%Display Final Intercept Data
disp('INTERCEPTION 2');
disp (['Target_2 Range =' num2str(distT_2 / 1000) ' km.'])
disp (['Missile_2 Range =' num2str(distM_2 / 1000) ' km.'])
disp (' ' );
disp (['Intercept Time_2 =' num2str((max(tArray_2)-lTimeT_2)/60) ' minutes.'])
disp (['Miss Distance =' num2str(min(distArrayMT_2)) ' m.'])
disp (['Lateral Divert =' num2str(max(latDivArrayM_2)) ' m/s.'])
disp (' ');
disp ('Simulation Finished.');
toc;                    % Calculate Run time
```

B. GEO2CART ( )

```
function r = geo2cart(strLatitude, strLongitude, R)
% GEO2CART      This will map the geographic coordinate system to Cartesian
%       coordinate sysytem.
%       This convert the geographical coordinate to the Cartesian
%       coordinate given that the position on the earth defined as
%       longitude an the latitude and earth is the sphere with radius R.
%
%       Examples of recognized formats are
%       123°30'00”S, 123-30-00S, 123d30m00sS and 1233000S.

% Copyright (c) 2004-2005 by Kursad YILDIZ

lat = deg2rad(npi2pi(str2angle(strLatitude)));
lon = deg2rad(npi2pi(str2angle(strLongitude)));
theta = pi/2-lat;
phi = lon;
x = R * sin(theta) * cos(phi);
y = R * sin(theta) * sin(phi);
z = R * cos(theta);
r = [x;y;z];
```

C. TOP2CART ( )

```
function y = top2cart(az, el, strLat, strLon)

% TOP2CART  This find the initial velocity unit vector in
%      Cartesian coordinate system of the ballistic missile
%      given that the geographic position and the azimuth and
%      elevation angle of the launch.
%
%       Examples of recognized formats are
%       123°30'00”S, 123-30-00S, 123d30m00sS and 1233000S.
```

```
%
%      This will return the velocity unit vector in the form
%      of Vu = [Vx;Vy;Vz]

% Copyright (c) 2004-2005 by Kursad YILDIZ

lat = deg2rad(npi2pi(str2angle(strLat)));
lon = deg2rad(npi2pi(str2angle(strLon)));

%Local transformation (R is assumed to be unity)
HA = sin(el);
EA = cos(el)*cos(az);
NA = cos(el)*sin(az);

%Global transformation
T = [-sin(lat)*cos(lon) -sin(lon) cos(lat)*cos(lon)
     -sin(lat)*sin(lon) cos(lon)  cos(lat)*sin(lon)
     cos(lat)          0         sin(lat)]; %Rotation Vector

y = T * [EA;NA;HA];
```

D. REFORMDATAMATRIX ( )

```
function y = reformDataMatrix(dataMatrix)
% REFORMDATAMATRIX    This add three new Rows and convert the pound weight
% to the Kg to the missile data matrix
%
%   reformDataMatrix(dataMatrix) is add dM/dt and canister Weight rows and
%   total weight to the data matrix. Before adding value it convert first
%   two row to Kg

%   Copyright (c) 2004-2005 by Kursad YILDIZ

numStage = size(dataMatrix, 2);   %Number of Stages
mcf = 0.4535924;    %Mass Conversion Factor (lb --> kg)
%Masses lb --> kg
for r = 1:2
    for c = 1:numStage
        dataMatrix(r, c) = dataMatrix(r, c) * mcf;
    end
end

% Add dM/dt and Canister Weight Rows to Data Matrix
intermVar1 = [];        % Define an intermediate variable
intermVar2 = [];        % Define an intermediate variable
for i = 1:numStage      % Loop for the number of stages
    intermVar1 = [intermVar1 (dataMatrix(2,i) /...
                  dataMatrix(4,i))];    % Generate dM/dt row
    intermVar2 = [intermVar2 (dataMatrix(1,i) -...
                  dataMatrix(2,i))];    % Generate canister weight row
end
dataMatrix = [dataMatrix; intermVar1; intermVar2];    % Reform Matrix

%Add Ignition Time Row to Data Matrix
intermVar1 = cumsum(dataMatrix(4,:));                  % Sum burn times
intermVar1 = [0 intermVar1(1:(size(intermVar1,2)-1))]; % Ignition time row
% Generate initial total mass
m1 = sum(dataMatrix(1,:));                      % Stage #1 Total weight
m2 = sum(dataMatrix(1,2:4));                    % Stage #2 Total weight
m3 = sum(dataMatrix(1,3:4));                    % Stage #3 Total weight
m4 = dataMatrix(1,4);                           % Stage #4 Total weight
intermVar3 = [m1 m2 m3 m4];

y = [dataMatrix; intermVar1; [m1 m2 m3 m4]];    % Reform Matrix
```

E. MAGNITUDE ( )

```
function y = magnitude(x)
% MAGNITUDE     This finds out the magnitude of the any Cartesian
%   coordinate vector which is extended to point (x,y,z) from origin
%   y = sqrt(x(1)^2 + x(2)^2 + x(3)^2...+ x(n)^2);
```

```
%
%   This return the found magnitude

%    Copyright (c) 2004-2005 by Kursad YILDIZ

y = sqrt(x'*x);
```

F. MOVEICBM ( )

```
function [F, state, altitude, Weight, pos, V, a, g, trc, old_trc, dist, dataMatrix] =...
            moveICBM(dt, state, dataMatrix, stage, old_trc, dist)
% MOVEICBM          This function find out the next state of ICBM
%       given prior state, dt dataMatrix, and its stage.
%
% This returns new state column matrix, altitude, weight and the missile
% and Gravitational Acceleration, distance from the launch site and the
% ground track of the ICBM

%   Copyright (c) 2004-2005 by Kursad YILDIZ

Re = 6.37e6;                    % Radius of the Earth (m)
G = 6.67e-11;                   % Gravitational Constant (m^3/s^2.kg)
Me = 5.98e24;                   % Earth's Mass (kg)
Cd = 1.25;                      % missile drag coefficient
Area = pi* 1.30^2;              % Cross sectional area meter square
Hp = [1 0 0 0 0 0;
      0 1 0 0 0 0;
      0 0 1 0 0 0];             % The position observation matrix
Hv = [0 0 0 1 0 0;
      0 0 0 0 1 0;
      0 0 0 0 0 1];             % Velocity observation matrix

dMdt =  dataMatrix(5, stage);   % Fuel burn ratio
ISP  =  dataMatrix(3, stage);   % Specific Impulse
M    =  dataMatrix(8, stage);   % Current mass
pos = Hp*state;                 % Current position
V = Hv*state;                   % Current velocity
magX =  magnitude(pos);         % Magnitude of position vector
magV =  magnitude(V);           % Magnitude of velocity vector
unitX = unitVector(pos);        % Position Unit vector
unitV = unitVector(V);          % Velocity Unit vector
altitudeOLD = magX - Re;        % Current Altitude of the missile (m)
g = (G * Me) / (magX ^ 2);      % Gravitational Acceleration (g)

% Create the transition Matrix for ICBM
W = G * Me / magX^3;            % Weight
Tr = dMdt * ISP * G * Me /...
     (magX^2 * magV * M);       % Trust
ro = rho(altitudeOLD);          % Air density
Dr = ro * G * Me * Cd * Area/...
          (2 * M * magX^2 );    % Drag

a = (Tr-Dr)*V - pos*W;          % Acceleration

T1 =[0       0          0          1         0          0;
     0       0          0          0         1          0;
     0       0          0          0         0          1;
     -W      0          0          Tr-Dr     0          0;
     0      -W          0          0         Tr-Dr      0;
     0       0         -W          0         0          Tr-Dr];
F = eye(6) + T1*dt;

% Move the missile to next position
state = F * state;

altitude = magnitude(Hp*state)-Re; % Altitude of the missile (m)
Weight = M - dMdt * dt;            % Reduce total weight
dataMatrix(8, stage) = Weight;     % Store the weight in dataMatrix
pos = Hp*state;                    % Next Position
V = Hv*state;                      % Next Velocity
trc = unitVector(pos) * Re;        % Ground track Vector
```

```
dist = dist + magnitude(trc - old_trc); % Range from launch site
old_trc = trc;
```

G. UNITVECTOR ( )

```
function y = unitVector(R)
% UNITVECTOR        This find out the unit vector of the Cartesian
%   coordinate vector R which is extended to point (x,y,z) from origin
%
%   Returns the unit vector of the given Vector

%   Copyright (c) 2004-2005 by Kursad YILDIZ
if magnitude(R) == 0
    y = [0;0;0];
else
    y = R / magnitude(R);
end
```

H. RHO ( )

```
function y = rho(altitude)

% RHO       This calculate the air density for given altitude. This
%       use the exponential approximation for the air density
%
%       Altitude, for which we need to find the air density, is in meter.

%   Copyright (c) 2004-2005 by Kursad YILDIZ


ro1 = 0.002378;     % Atmospheric density constant-1  (altitude < 30000 ft)
ro2 = 0.0034;       % Atmospheric density constant-1  (altitude > 30000 ft)
Kp1 = 30000;        % Atmospheric Constant-2  (altitude < 30000 ft)
Kp2 = 22000;        % Atmospheric Constant-2  (altitude > 30000 ft)
ft2m = 0.3048;      % Conversion coefficient from ft to meter
den_con = 515.1836; % slug/ft^3  ----> kg/m^3 conversation coefficient


if (altitude <= 30000*ft2m)
    y = den_con * ro1*exp(-altitude/(ft2m*Kp1)); % Air density (kg/m^3)
else
    y = den_con * ro2*exp(-altitude/(ft2m*Kp2)); % Air density (kg/m^3)
end
```

I. MOVEDECOYS ( )

```
function [state, altitude, pos, V, trc, old_trc, dist, dist2] =...
          moveDecoys(dt, state, pM, dist, old_trc)
% MOVEDECOY     This function finds out the next state of Decoy,
%       given earlier state position and dt time interval.
%
% This returns new state column matrix, altitude, position, velocity, ground
% track, range and missile decoys separation

%   Copyright (c) 2004-2005 by Kursad YILDIZ


Re = 6.37e6;                    % Radius of the Earth (m)
G = 6.67e-11;                   % Gravitational Constant (m^3/s^2.kg)
Me = 5.98e24;                   % Earth's Mass (kg)
Cd = 1.25;                      % Missile drag coefficient
Area = pi* 0.5^2;               % Decoys cross sectional area meter square
M = 90;                         % Decoys weight (kg)
Hp = [1 0 0 0 0 0;
    0 1 0 0 0 0;
    0 0 1 0 0 0];               % The position observation matrix
Hv = [0 0 0 1 0 0;
    0 0 0 0 1 0;
    0 0 0 0 0 1];               % velocity observation matrix
pos = Hp*state;                 % Current position
```

```
magX = magnitude(pos);          % Magnitude of Position

altitudeOLD = magX - Re;        % Altitude of the missile (m)

% create the transition Matrix for ICBM
W = G * Me / magX^3;            % Weight
Tr = 0;       % Trust
ro = rho(altitudeOLD);          % Air Density
Dr = ro * G * Me * Cd * Area/...
    (2 * M * magX^2 );          % Drag

T1 =[0       0           0           1          0           0;
    0       0           0           0          1           0;
    0       0           0           0          0           1;
    -W      0           0           Tr-Dr      0           0;
    0      -W           0           0          Tr-Dr       0;
    0       0          -W           0          0           Tr-Dr];
T = eye(6) + T1*dt;
% find the new state-space of the missile
state = T * state;
altitude = magnitude(Hp*state)-Re; % Altitude of the missile (m)
pos = Hp*state;
V = Hv*state;
%Integrate decoy downrange
trc = UnitVector(pos) * Re;
dist = dist + magnitude(trc - old_trc);
old_trc = trc;
% Missile -decoy distance
dist2 = magnitude(pM - pos);
```

J. SCOPE ( )

```
function [measurements, covariance] = Scope(isDecoys, isReduced, isJamming, targets, de-
coys, posRF)

% SCOPE    In multi-target multi-sensor environment, this simulates
%           the radars sensing ability.
%
%           A stack of radar measurement and related stack of covariance
%           matrix, which correspond the clutter, false target and the real
%           target echoes will be returned.
%
%           If the any of the expandable decoys is released then the
%           measurement stack will include it.
%
%           if the reduced RCS is used then the radar error will increase.
%           Because the value of the RCS do not change significantly and
%           The simulation has already used the interpolation for the
%           detailed degrees. This will just use stage one RCS value for
%           standard deviation.
%
%           if the jamming is on then radars returns to angle only
%           measurement.
%
%           isDecoys is the flag array that if the any of the expandable
%               decoys are released or not. Depending on the number of
%               targets out there the length of the array will increase. If
%               any point in the array is different then “zero”, then
%               related index numbered target release the decoy.
%           isReduced is the flag that indicates the use of Reduced RCS
%           isJamming is the flag that indicates the changing angle only
%               measurement
%           targets is the stack that contains the current state of the
%               all targets out in the sky in Cartesian coordinate.
%           posRF is the  Radar position in Cartesian coordinate
%
%           See also sensePositionRF, senposition

% Copyright (c) 2004-2005 by Kursad YILDIZ

global RCS1X RCS1X_R
```

```
%% Constants
% This assumes that all the radars use the same parameter. If the radar
% parameter will be changed construct a “switch” to assign the parameter
%       = [1          2             3          4        5          6
%       7    8   9   10        11     12                13]
% RdPar = [frequency, 3dB_Bandwith, Peak_Power, Gain,    Pulsewith,
% Rcvr_Bandwith, PRF, Ni, Fr, lambda,   km,    3dB_band_in_radin, FrBs];
RdPar = [10e9 0.5 1000e3 104000 50e-6 20000 150 20 4 0.03 1.7 0.0087266 20];

SOL = 3e8;          % Speed of light (m/s)
kT0 = 4e-21;        % Boltzmann Constant x 290K
RCSD = 10^(5/10);   % Decoy Radar Cross Section (5 dBsm)
Pm = 0.001;         % Probability of miss while target exist

Hp = [1 0 0 0 0 0;
      0 1 0 0 0 0;
      0 0 1 0 0 0];                         % Position Observation Matrix
Hv = [0 0 0 1 0 0;
      0 0 0 0 1 0;
      0 0 0 0 0 1];                         % Velocity observation Matrix
% Create false target/clutter for this radar
numfalse = 5;                               % The number of false target or
                                            % clutter created by each radar
                                            % is assumed to be 10 per scan
                                            % per validation volume.
%% Prepare for measurements
if isReduced
    RCSindex = RCS1X_R;           % Reduced RCS look-up table
    snr = 10;
else
    RCSindex = RCS1X;             % Normal RCS look-up table
    snr = 500;
end

measurements = [];                          % Empty Measurements stack
covariance = [];                            % Empty covariance stack

for i = 1:size(targets,2)
%% Make the target measurements
    pos = Hp*targets(:,i);                  % Related target true Position
    vel = Hv*targets(:,i);                  % Related target true velocity
    LOS = pos - posRF;                      % Line of Sight
    Range = magnitude(LOS);                 % Magnitude of LOS
    lookAngle = acos(dot(unitVector(LOS),...
                         unitVector(vel))); % Radar Look Angle (rad)

    %Determine RCS Seen by RF Sensors
    RCS = RCSindex(round(rad2deg(lookAngle)*10) + 1);
    RCS = 10 ^ (RCS / 10);                  %Convert RCS values : dBsm > sm
    SNR = (RdPar(3)* RdPar(4)^2*RdPar(5) *1*RCS*RdPar(10)^2)/...
        ((4*pi)^3*kT0*RdPar(9)*Range^4);    % Calculate SNR
    sigmaA = RdPar(12)/(RdPar(11)*...
        sqrt(2*snr*RdPar(13)));             % Angular Error Std. Dev. (rad)
    sigmaR = (SOL*RdPar(5)/2)/(RdPar(11)*...
        sqrt(2*snr*RdPar(13)));             % Range Error Std. Dev. (m)

    % Convert LOS to the spherical coordinate
    [tt,pp,rr] = cart2sph(LOS(1),LOS(2),LOS(3));    % Convert to spherical
    Zs = [rr;tt;pp];
    sigma = [sigmaR;sigmaA;sigmaA];                 % STD.DEV vector
    cov = diag(sigma.^2);                           % Measurement covariance
    % If the target use barrage jamming then switch angle only
    if isJamming
        Zs(1) = 0;                                      % No range information
    end % end for is jamming for targets
    Zsm = Zs + randn(size(sigma)).*sigma;               % Add the error
    [xx,yy,zz] = sph2cart(Zsm(2), Zsm(3), Zsm(1));
    Zcm = posRF + [xx;yy;zz];                           % Convert to Cartesian to find
err
    err = magnitude(pos - Zcm);                         % Find error in measurement
```

```
    if rand < Pm
        Zsm = [0;0;0];                                  % Radar may miss the target
    end
    Zsm = [Zsm; posRF];                                 % Add RF position for evaluation
    measurements = [measurements Zsm];                  % Add to the stack
    covariance = [covariance cov];                      % Add to the stack

%%  Make the measuremnts for the False targets and clutters
    co_sigma = (1 + 1*rand(1,numfalse));    % False target can be placed
                                            % 1*sigma to 1*sigma window
                                            % from the target. The longer
                                            % distance is already gated out
                                            % by assumption
    for k = 1:numfalse
        sigmaF = sigma*co_sigma(k);         % Sigma for False target
        Zsm = Zs + randn(size(sigma)).*sigma;% Add the error
        Zsm = [Zsm; posRF];                 % Add RF position for evaluation
        measurements = [measurements Zsm];
        covariance = [covariance cov];
    end % end for false target and clutter

%%  Take the measurement for only the decoy which is released
    if isDecoys(i)
        posD = Hp*decoys(:,i);              % Current position of decoy
        LOSD = posD - posRF;                % Line of Sight
        SNR = (RdPar(3)*RdPar(4)^2*RdPar(5) *1*RCSD*RdPar(10)^2)/...
            ((4*pi)^3*kT0*RdPar(9)*magnitude(LOSD)^4);      % Calculate SNR
        sigmaAD = RdPar(12)/(RdPar(11)*...
            sqrt(2*SNR*RdPar(13)));         % Angular Error Std. Dev. (rad)
        sigmaRD = (SOL*RdPar(5)/2)/(RdPar(11)*...
            sqrt(2*SNR*RdPar(13)));         % Range Error Std. Dev. (m)
        [tt,pp,rr] = cart2sph(LOSD(1),LOSD(2),LOS(3));    % Convert to spherical
        Zs = [rr;tt;pp];
        sigmaD = [sigmaRD;sigmaAD;sigmaAD];            % STD.DEV vector
        % Convert LOS to the spherical coordinate
        if isJamming
            Zs(1) = 0;                                      % No range information
%             sigmaD(1) = 0;                                  % STD.DEV vector
        end % end for is jamming
        Zsm = Zs + randn(size(sigmaD)).*sigmaD;              % Add the error
        Zsm = [Zsm; posRF];                                 % Add RF position for evalua-
tion
        measurements = [measurements Zsm];                  % Add to the stack
        covariance = [covariance cov];                      % Add to the stack
    end % end for if isDecoys
%%
end    % end for outer for loop

%% Use the same column number
% Be sure all measurement stacks have the same number of columns. There
% should be 4 more measurements than number of false target/clutter. Two
% for real targets, two for decoys (Not necessarily but needed to have the
% same number columns)
nubcol = size(targets,2)*numfalse + size(targets,2) + size(decoys,2);% Max number of col-
umns
add = nubcol - size(measurements,2);        % Needed extra columns
for j = 1:add
    measurements = [measurements zeros(6,1)];   % Fill with zeros
    covariance = [covariance zeros(3)];         % Fill with zeros
end
```

K. MASH ( )

```
function [states, cov] = Mash(isJamming, PreStates, PreCov, Fs, measurements, covari-
ances)

% MASH In the multi-target multi-sensor environment this simulates the
%       measurements correlation, association, and Kalman filtering.
%
%       The inputs are array of measurements and covariance. This will apply
%       the ellipsoidal gating to correlate the measurement with the track. If
```

```
%       the resulting correlation has more than one measurements than
%       this will apply the probabilistic analyze to associate the track.
%
%       The track initiation is done by the help of the IR sensor which is
%       not simulated in the original simulator. The new track will send to
%       here by “ad hoc” and test will apply if any measurements correlate
%       and associate with this new information.
%
%       isJamming is the indication of angle only measurement
%       PreStates is the stack of previous states of the track
%       PreCov is the array of previous states covariance
%       Fs is the array of transition matrix
%       measurements is the array of measurements that comes from the
%       RF sensors
%       covariances is the array of the measurement covariance
%
%       states is the array of Kalman filtered new states of previously
%       inilized track
%       cov is the array of new covariance of previously initialized track
%
%       See also fusionBox

% Copyright (c) 2004-2005 by Kursad YILDIZ

warning('off');

%% Constants
Hm = [eye(3),zeros(3)];            % Measurement Observation Matrix
Hrf = [zeros(3), eye(3)];          % RF Position observation Matrix
Pd = 0.999;                        % Probability of detection
Pg = 0.9786;                       % Probability of target within correlation window
m = 5;                             % Expected number of measurement in the gate volume
M = 3;                             % Measurement dimension

% Gate threshold that the probability of mass Pg in side the gate is 0.9786.
gama = 9;                           % 4sigma window

dt = 0.05;                              % Fusion box time interval (s)
q2 = 0.001;                             % Plant Noise Coefficient square = 0.001
Q = q2 * [dt^3/3   0       0      dt^2/2      0       0;
          0       dt^3/3  0       0           dt^2/2  0;
          0       0       dt^3/3  0           0       dt^2/2;
          dt^2/2  0       0       dt          0       0;
          0       dt^2/2  0       0           dt      0;
          0       0       dt^2/2  0           0       dt];

%% Corralate, Associate, Fuse start-up
states = [];                    % Empty states vector array
cov = [];                       % Empty Covariance matrix Array
ii = 1;                         % Outer for matrix index
iend = size(PreStates,2);       % number of target

for i = 1:iend

%% PREDICTION Phase For Kalman Filter
    F = Fs(1:6,ii:ii+5);        % Related Transition
    x = PreStates(:,i);         % Related state vector
    P = PreCov(1:6, ii:ii+5);   % Related covariance
    ii = ii + 6;                % Increase for next step
    x_k = F*x;                  % Predicted position
    P_k = F*P*F' + Q;           % Prediction covariance

%% CORRELATION
    track = [];                 % Empty Track information array for related target
    trackP = [];                % Empty track covariance array for related target
    Tr = [];                    % Empty trace of track covariance array
    kk = 1;                     % Zs  matrix index
    kkk = 1;                    % Rs matrix index
    % There is 6 rows, 3 for measurement, 3 for RF position
    kend = (size(measurements,1)/6);    % Number of radars
    for k = 1:kend
```

```
        % Select stack to test
        Zs = measurements(kk:kk+5, 1:size(measurements,2));   % Related measurements
        Rs = covariances(kkk:kkk+2, 1:size(covariances,2));   % Related covariance
        LOS = Hm*x_k - Hrf*Zs(:,1);                           % LOS OF RELATED RF

        [tt,pp,rr] = cart2sph(LOS(1), LOS(2), LOS(3));        % Predicted measurement
        h = [rr;tt;pp];
        % Measurement matrix is the gradient of the measurement function
        H = Hk(LOS);
        % Index increasing
        kk = kk + 6;                % Increase for next step
        kkk = kkk + 3;              % Increase for next step

        % Find the Measurement Matrix ('h' and 'H')for extended Kalman filter
        if isJamming
            h(1) = 0;               % No range observation
            H(1,1:6) = zeros(1,6);
        end % End for if isjamming

         % 'R' and 'S' is the same for within radar measurements
        R = Rs(1:3, 1:3);           % R  measurement covariance
        S = H*P_k*H' + R;           % Residual covariance

        AsoMat =[];                 % Empty correlated innovation matrix
        ej = [];                    % Empty probability matrix
        Darray = [];                % Empty statistical distance array
        CorrD = [];                 % Correlated statistical distance
        Corrmu = [];                % Correlated innovation
        Marray = [];                % Test array
        nend = size(Zs,2);          % Number of observation to test
        for n = 1:nend
            z = Hm*Zs(:,n);         % Measurement to test
            mu = z - h;             % Residual
            D = abs(mu'*inv(S)*mu); % Statistical distance
            Darray = [Darray D];
            AsoMat = [AsoMat mu];   % Correlated innovation
            if D <= gama            % Test against window
                CorrD = [CorrD D];
                Corrmu = [Corrmu mu];
            end
        end % End for measurements loop

        if length(CorrD) > 0 % Use at most 5 data
            if length(CorrD) > 4
                Marray = Corrmu(:,1:3);
                ej = exp(-1/2*CorrD(1:3));
            else
                Marray = Corrmu;
                ej = exp(-1/2*CorrD);
            end
        else
            for n = 1:4                         % Just pick up m to use
                [D index] = min(Darray);
                ej = [ej (exp(-1/2*D))];         % Statistical distance probability
                Marray = [Marray AsoMat(:,index)];% Store the first min 5
                Darray(index) = max(Darray);     % Do not choose again
            end
        end

%% ASSOCIATION WITH JPDA
        jend = size(Marray,2);  % Number of correlated measurement

        % Find Combined Residual weighted with own probability
        c = sum(ej);            % Normalizing Constant
        b = m*sqrt(2*pi)*(1 - Pd*Pg)/...
                 (gama*pi*Pd);  % Probability miss coefficient
        Betanot = b/(b + c);    % Probability that there is no target return in this
stack
        Betaj = ej./(b + c);    % Each correlation probability
        muT = [0;0;0];                % Combined innovation
```

```
        for j = 1:jend
            muT = muT + Betaj(j)* Marray(:,j);   % Combined innovation
        end

%% CORRECTION Phase for Kalman Filtering
        % Find the covariance increase coefficient
        sumP = 0;
        for j = 1:jend
            sumP = sumP + (Betaj(j)*Marray(:,j)*(Marray(:,j))' - muT*muT');
        end
        K      = P_k*H'*inv(S);           % Filter gain
        % Update covariance
        P_c    = (eye(length(K)) - K*H)*P_k*(eye(length(K)) - K*H)' + K*R*K';
        % Incrase in covariance due to use of multiple measurement
        Phat   = K*sumP*K';
        P_post = Betanot*P_k + (1 - Betanot)*P_c + Phat;    % New Covariance
        xhat   = x_k + K*muT;

%% Store track information
        track = [track xhat];
        trackP = [trackP P_post];
        Tr = [Tr abs(trace(P_post))];
    end % end for radar loop

%% Fuse the track files
    c = sum(1./Tr);                    % Normalizing Constant
    st = 0;
    pp = 0;
    qq = 1;                           % Index for choosing covariance
    qend = size(track,2);             % Number of track file to fuse
    for q = 1:qend
        st = st + (1/Tr(q))/c*track(:,q);
        pp = pp + (1/Tr(q))/c*trackP(1:6,qq:qq+5);
        qq = qq + 6;
    end % End for Fusion

%% Store the results for related target
    states = [states st];
    cov = [cov pp];
end % end for states loop
```

L. HK ( )

```
function H = Hk(R);
% HK        This will find the measurement matrix for the extended kalman
%           filter. For this EKF, non-linearity is for the measurement such
%           that the sensor observe the [range;azimuth;elevation];
%
%           The measurement function is assumed to be as z = [sqrt(x^2 +
%           y^2 +z^2);
%                atan2(y,x); atan2(z,sqrt(x^2 + y^2))]
%
%           This will return the measurement matrix

% Copyright (c) 2004-2005 by Kursad YILDIZ

x = R(1);
y = R(2);
z = R(3);
a1 = sqrt(x^2 + y^2 + z^2);
a2 = sqrt(x^2 + y^2);
H = [x/a1          y/a1            z/a1     0    0    0;
    -y/a2^2        x/a2^2          0        0    0    0;
    -z*x/(a2*a1^2)     -z*y/a2/a1^2       a2/a1^2      0    0    0];
```

M. MOVEINTERCEPTOR ( )

```
function [state, dataMatrix, pos, V, unitV, g, a, alt, M, dist, grd_trc, grd_trc_old]
=...
         moveInterceptor(dt, state, dataMatrix, stage, GF, grd_trc_old, dist)
```

```
% MOVEINTERCEPTOR move forward the given ICBM INTERCEPTOR for the next step
%
%          dt is the time interval of the simulation state is the current
%          state of the ICBM dataMatrix is the data for the ICBM stage is
%          the stage number which the ICBM is in GF is the guidance force
%          that need to turn the ICBM
%
%          this returns state is new state vector of the ICBM datamatrix is
%          new data matrix that the total weight is reduced g is the
%          gravitational acceleration needed for the guidance alt is the
%          current altitude of the ICBM

% Copyright (c) 2004-2005 by Kursad YILDIZ


Re      = 6.37e6;          % Radius of the Earth (m)
G       = 6.67e-11;        % Gravitational Constant (m^3/s^2.kg)
Me      = 5.98e24;         % Earth's Mass (kg)
Cd      = 1.25;            % missile drag coefficient
Area    = pi* 1.30^2;      % missile cross sectional area meter square
Hp = [1 0 0 0 0 0;
      0 1 0 0 0 0;
      0 0 1 0 0 0];        % The position observation matrix
Hv = [0 0 0 1 0 0;
      0 0 0 0 1 0;
      0 0 0 0 0 1];        % velocity observation matrix

dmdt =  dataMatrix(5, stage);   % Fuel burn ratio
Isp  =  dataMatrix(3, stage);   % Specific Impulse
M    =  dataMatrix(8, stage);   % State mass
if M <= 0       % Do not use fuel that is not in the tank
    dmdt = 0;
end
pos = Hp*state;                 % Current Position
V = Hv*state;                   % Current Velocity
magX =  magnitude(pos);         % Magnitude of position vector
magV =  magnitude(V);           % Magnitude of velocity vector
% Find altitute
alt = magX - Re;                % Current altitude
g = (G * Me) / (magX ^ 2);      % Gravitational Acceleration (g)
W = G * Me / magX^3;            % Weight
Tr = dmdt * Isp * G * Me /...
    (magX^2);                   % Trust
magGF = magnitude(GF);          % Magnitude of guidance force

% Compensate for Guidance Command
% If Lateral Acceleration Requirements Exceeds Available Thrust
if (magGF >= Tr)
    magGF = Tr;                 % Apply as Much as Available
    Tr = 0;                     % Set Thrust to zero
else % If not, compensate Thrust for the Guidance Force
    Tr = sqrt(Tr^2 - magGF^2);
end
Tr = Tr/(magV * M);             % Magnitude of the applicable Thrust
ro = rho(alt);                  % Air density
Dr = ro * G * Me * Cd * Area /...
    (2 * M * magX^2 );          % Drag
a = (Tr-Dr)*V -...
     pos*W;                % Acceleration
T1 =[0       0          0          1         0          0;
     0       0          0          0         1          0;
     0       0          0          0         0          1;
     -W      0          0          Tr-Dr     0          0;
     0      -W          0          0         Tr-Dr      0;
     0       0         -W          0         0          Tr-Dr];
T2 = [0;0;0;GF];                % Guidance force in 6 dimension
F = eye(6) + T1*dt;
state = F*state + T2*dt/M;      % Next step
pos = Hp*state;                 % Next Position
unitX = unitVector(pos);        % Next position Unit Vector
```

```
V = Hv*state;                   % Next Velocity
unitV = unitVector(V);          % Next velocity Unit Vector
M = M - dmdt * dt;              % Reduce total weight
dataMatrix(8, stage) = M;
alt = magnitude(pos) - Re;      % New altitude
%Integrate missile downrange
grd_trc = unitX * Re;           % Ground track Vector
currDistM = grd_trc - grd_trc_old;
dist = dist + magnitude(currDistM);
grd_trc_old = grd_trc;          % Record previous ground track
```

N. GUIDANCE ( )

```
function [distMT, VcMTTrue, oldDistTrue, Vc, oldDist, oldLOS, mnvr, Nlm, GFM,...
          magGFM, magNC1, comLatAccM, achLatAccM, LateralDiv] =...
            guidance(which, TargetX, MissileX, sensedPos, Velocity, Vc, lounchT,...
                     oldDistTrue, oldDist, oldLOS, a, gICMB, Nlm, g, Weight, unitV,...
                     LateralDiv, magNC0)

% GUIDANCE      This will calculate the guidance force and all the other
%       potable values for the missile interception
%
%       which indicates the interception number
%       TargetX true position of the target
%       MiiisleX true position of the interceptor
%       sensedPos is the estimated state of the target
%       Velocity true velocity of the target
%       Vc is the prior Closing velocity
%       LounchT is the launch time of the target
%       oldDistTrue is the true prior target ---> Interceptor distance
%       oldDist is the estimated target ---> Interceptor distance
%       oldLOS is the prior line of sight target--->Interceptor
%       a is the true acceleration of the target
%       gICBM is the true gravitational acceleration of the target
%       Nlm is prior achieved acceleration
%       g is the gravitational acceleration of the interceptor
%       Weight is mass of interceptor
%       UnitV is velocity unit vector of interceptor
%       LateralDiv is previous lateral divert
%       magNCO is magnitude of previous commanded acceleration
%
%       This returns
%       distMT is the target---> Interceptor distance
%       VcMTTrue is true Closing velocity
%       oldDistTrue is the next old distance true
%       Vc is the current Closing velocity
%       oldDist is the next old distance
%       oldLOS is the Next old line of sight
%       mnvr is the target maneuver
%       Nlm is the current achieved acceleration
%       GFM is the current guidance force
%       magGFM is the magnitude of the guidance force
%       magNC1 is the magnitude of the commanded acceleration
%       comLatAccM is the current commanded lateral acceleration magnitude
%       achLatAccM is the current achieved acceleration magnitude
%       LateralDiv is the current lateral divert of the missile

%   Copyright (c) 2004-2005 by Kursad YILDIZ

global timeFlags updateTime navCoefM Vcfirst SSM1 SSM2 stateM1 stateM2...
       maxG TMc;

switch which
    case 1
        time = timeFlags(1,1);
        deltaT = timeFlags(2,1);
        txF = timeFlags(3,1);
        txC = timeFlags(4,1);
        updateT = updateTime(1);
        txD = timeFlags(5,1);
        navCoef = navCoefM(1);
```

```
        stateMX =  stateM1(:,1);
        stateMY =  stateM1(:,2);
        stateMZ =  stateM1(:,3);
        AMc = SSM1(:,1:3);
        BMc = SSM1(:,4);
        CMc = SSM1(:,5)';
        DMc = SSM1(1,5);
        mG = maxG(1);
    case 2
        time = timeFlags(1,2);
        deltaT = timeFlags(2,2);
        txF = timeFlags(3,2);
        txC = timeFlags(4,2);
        updateT = updateTime(2);
        txD = timeFlags(5,2);
        navCoef = navCoefM(2);
        stateMX =  stateM2(:,1);
        stateMY =  stateM2(:,2);
        stateMZ =  stateM2(:,3);
        AMc = SSM2(:,1:3);
        BMc = SSM2(:,4);
        CMc = SSM2(:,5)';
        DMc = SSM2(1,5);
        mG = maxG(2);
    otherwise
        updateT = 0.15;
end
% Initial Values
VcMTTrue = 0;
mnvr = 0;
GFM = [0;0;0];
magGFM = 0;
comLatAccM = 0;
achLatAccM = 0;
ncM = [0;0;0];
magNC1 = magNC0;
% -----------------------------------------------------------------
LOSMTTrue = TargetX - MissileX;             %True Target-Missile Vector
distMTTrue = magnitude(LOSMTTrue);          %True Target-Missile Distance

%Apply transmission delay
receivedPosT = sensedPos + (-Velocity .* txD);

LOSMT = receivedPosT - MissileX;    %Line of Sight (LOS) Between Missile and Target
distMT = magnitude(LOSMT);          %Target-Missile Distance

if time > (deltaT + lounchT)
    VcMTTrue = (oldDistTrue - distMTTrue) / deltaT;             %True Closing Velocity
(Vc)
    %Compute Control Acceleration at Only Data Update Intervals
    if txF || (txC >= updateT)
        % For the fist time of calculation we need 0 for Vc
        switch which
            case 1
                if Vcfirst(1);
                    oldDist = distMT;
                    Vcfirst(1) = 0; % Never set Vc 0 again
                end
            case 2
                if Vcfirst(2);
                    oldDist = distMT;
                    Vcfirst(2) = 0; % Never set Vc 0 again
                end
        end
        Vc = (oldDist - distMT) / updateT;                      % Closing Velocity (Vc)
        %Compute Magnitude and Direction of Lateral Acceleration
        unitOldLOS = unitVector(oldLOS);                        % Normalize Previous LOS
        unitLOSMT = unitVector(LOSMT);                          % Normalize This LOS
        deltaLOS = unitLOSMT - unitOldLOS; % Find LOS Change Direction (=Direction of
Lateral Acceleration)
        magLOSRate = magnitude(deltaLOS) / updateT;       % Magnitude of LOS Rate (rad/s)
```

```
        unitncM = unitVector(deltaLOS);                 % Lateral Acceleration Unit
Vector
        magNC1 = navCoef * magLOSRate * Vc;  % Magnitude of Lateral Acceleration (m/s^2)
        ncM = magNC1 * unitncM;              % Lateral Acceleration Vector (m/s^2)
        if time <= 90
            Vc = abs(Vc);   % Never allow before real interception
        end
        %Reset counters/flags
        switch which
            case 1
                timeFlags(3,1) = 0;
                timeFlags(4,1) = 0;
            case 2
                timeFlags(3,2) = 0;
                timeFlags(4,2) = 0;
        end
        oldDist = distMT;
        oldLOS = LOSMT;
    end %txF | (txC >= updateT)
    %Compute Target Acceleration Perpendicular to LOS (Target Maneuver)
    magaT = magnitude(a);  % Magnitude of Target Acceleration (m/s^2)
    unitaT = a / magaT; %Target Acceleration Unit Vector
    alfa = acos(dot(unitaT, -unitVector(LOSMT))); %Angle Between Target Acceleration Vec-
tor and LOS (rad)
    magaPLOST = magaT * sin(alfa); %Target Acceleration Component Perpendicular to LOS
(m/s^2)
    mnvr = magaPLOST / gICMB; %Target Maneuver (g)
    if TMc == 0 %Control System Dynamics Implementation
        Nlm = ncM;
    else
        %Implement Control System Dynamics
        %x-axis
        nlMX = CMc * stateMX + DMc * ncM(1);
        stateMX = AMc * stateMX + BMc * ncM(1);
        %y-axis
        nlMY = CMc * stateMY + DMc * ncM(2);
        stateMY = AMc * stateMY + BMc * ncM(2);
        %z-axis
        nlMZ = CMc * stateMZ + DMc * ncM(3);
        stateMZ = AMc * stateMZ + BMc * ncM(3);
        switch which
            case 1
                stateM1 = [stateMX, stateMY, stateMZ];
            case 2
                stateM2 = [stateMX, stateMY, stateMZ];
        end
        Nlm = [nlMX; nlMY; nlMZ];  %Achieved lateral acceleration vector
        if Nlm == [0;0;0]
            Nlm = ncM;
        end
    end %TMc == 0 (Control System Dynamics Implementation)
    % Guidance Force
    magnlM = magnitude(Nlm);              %Magnitude of achieved lateral acceleration
    comLatAccM = magNC1 / g;              %Commanded Lateral Acceleration (g)
    achLatAccM = magnlM / g;              % Achieved Lateral Acceleration (g)
    if achLatAccM >= mG
        achLatAccM = mG;
        magnlM = mG*g;
        Nlm = magnlM*unitVector(Nlm);
    end
    LateralDiv = LateralDiv + abs(magnlM * deltaT);   %Lateral Divert (m/s)
    %Compute Lateral Acceleration Perpendicular to Velocity Vector, Ignore Parallel Com-
ponent
    nlPerM = Nlm - unitV * magnlM * cos(acos(dot(unitVector(Nlm), unitV)));   %Achieved
Acceleration Vector Perpendicular to Velocity Vector (Use This)

    GFM = nlPerM * Weight;                    % Guidance Force (N) (Perpendicular to LOS)

    magGFM= magnitude(GFM); %Magnitude of guidance force
end
oldDistTrue = distMTTrue; %Record old Missile-target distance.
```

APPENDIX C. READ-ME

This appendix contains the "read-me file" of the MATLAB® code given in Appendix B. The appendix is intended to help future users to easily modify the code.

To work with the simulation, all MATLAB® functions given in Appendix B and *POstage1_X.mat* and *RCS1X_R.mat* data files need to be copied to the same directory. After copying the files, type *MultiTarget3D* in the MATLAB® command window to run the simulation.

All variable names specified in the functions are an abbreviated form of the original names. The comment line next to each related variable explains the original long form of the variable. A "T" at the end of a variable refers to the target ICBM, an "M" refers to the interceptor ICBM, a "D" refers to a decoy, an "MT" refers to a variable that relates the target and the interceptor, and the "TD" refers to the variable that relates the decoy and the target. A "_1" or "_2" at the end of a variable refers to interceptor/target pairs. For this simulation, we assume that only two interceptions take place.

The simulation runs in the ECEF (Earth-Centric, Earth-Fixed) Cartesian coordinate system. All other coordinate variables should be converted to ECEF. The ICBM parameter includes the position, the launch angle, and the data matrix. The position of the ICBM, like all other positions defined in the simulation, is given in the geodetic coordinate system. The *longitude* and the *latitude* of the related positions are converted to the ECEF Cartesian coordinate system by using the *geo2cart.m*, which assumes that the earth is a perfect sphere. The launch angle of the ICBM is given in a topographic coordinate system, which assumes a local flat surface; *top2cart.m* converts this data to ECEF Cartesian coordinate system.

The format of the input data matrix is given in Table IV-8. *reformDataMatrix.m* converts this matrix to Table IV-9.

To change the parameters of the ICBM, the input data matrix needs to be changed. The position and the launch angle can be changed to any realistic value by modifying the related variable in the simulation.

To add more interception, only the *if-blocks* within the main *while-loop* need to be reintroduced. The new parameter should be distinguished by adding "_3" at the end of the variable. The *scope.m* and *mash.m* functions work autonomously. The *if-blocks* before these functions need to be modified to include newly added interception parameters.

LIST OF REFERENCES

[1] "Boeing ground based interceptor (GBI)", http://www.designation-systems.net/dusrm/app4/gbi.html, last accessed June 09, 2005

[2] D.C.Gompert, J.A.Isaacson, "Planning a ballistic missile defense system of system", RAND, National Defense Research Institute Issue paper, 1999

[3] "Missile Defense Progress Report", http://www.ceip.org/files/projects/npp/pdf/MissileDefenseProgressReport.pdf, last accessed June 09, 2005

[4] UCS/MIT report on "Countermeasures: A Technical Evaluation of the Operational Effectiveness of the Planned US National Missile Defense System Appendices A-E", April 2000

[5] Kubilay UZUN, "Requirements and limitations of boost–phase ballistic missile intercept systems", Master's Thesis, Naval Postgraduate School, Monterey, California, September 2004

[6] Gokhan Humali, "Sensor fusion for boost phase interception of ballistic missiles", Master's Thesis, Naval Postgraduate School, Monterey, California, 2004

[7] Merrill L. SKOLNIK, "*Introduction to radar systems*", Third edition, McGraw Hill, 2001

[8] David C. JENN, "*Radar and Laser cross section engineering*", AIAA,1995

[9] E. Garrido, "Graphical user interface for a physical optics radar cross section prediction code", Master's Thesis, Naval Postgraduate School, Monterey, California, 2000

[10] Personal conversation with Butch Caffall, Technical Director of MDA

[11] Richard N. Johnson, " Radar-Absorbing Material: A passive role in an active scenario", The international Countermeasure Handbook, 11th Edition, 2004

[12] Fawwaz T. ULABY, " *Fundamentals of applied Electromagnetic*", Prentice Hall, 2004

[13] Kemal YUZCELIK, "Radar absorbing material design", Master's Thesis, Naval Postgraduate School, Monterey, California, September 2003

[14] H.M. Musal, Jr., and D.C.Smith, "Universal Design chart for specular absorbers", IEEE Transaction on Magnetics, vol 26, no 5, September 1990

[15] Sh.V.MAMEDOV, Y.LENGER, S.BOLCAL, V.A ALEKPEROV, "Magnetodielectrical Polymer Composition Materials", TUBUTAK, 1990

[16] Emerson&Cumming, "Dielectric Material Chart", Emerson&Cumming, http://www.eccosorb.com/ Dielectric_Chart.pdf, last accessed June 09, 2005

[17] Ding SUN, "Measurement results of permittivity/permeability/loss Tangent of several microwave Absorbers", PBAR note 599, 1998, http://www-bdnew.fnal.gov/pbar/documents/pbarnotes/pdf_files/PB599.PDF , last accessed June 09, 2005

[18] Sergei A. VALKIN, Lev N. SHUSTOV, Robert H. DUNWELL, "*Fundemantals of Electronic Warfare*", Artech House, 2001

[19] D.Curtis SCHLEHER, "*Electronicwarfare in the information age*", Artech House, 1999

[20] Sherman W. MARCUS, "Dynamics and radar cross section density of chaff clouds", IEEE Transaction on Aerospace and Electronic Systems, Vol. 40, No. 1, January 2004

[21] T.A. WINCHESTER, "Pulsed radar return from a chaff cloud", IEEE Proceeding-F, vol. 139, No. 4, August 1992

[22] Phillip E. Pace, Notes for EC3700 *(*Introduction to Joint Services Electronic Warfare*)*, Naval Postgraduate School, 2004 (unpublished)

[23] American Physical Society Group's report on "Boost-phase intercept system for national system", 2003

[24] "Electronic warfare and radar systems engineering handbook", Naval Air Warfare Center, 1997

[25] F.NERI, "Anti-Monopulse jamming techniques", SBMO/IEEE MTT- SIMOC 2001 proceeding (invited paper), 2001

[26] T.Hun-Hau, "Effectiveness of off-board active decoys against anti-shipping missile" Master's Thesis, Naval Postgraduate School, Monterey, California, September 1996.

[27] A. Gelp et al, "*Aplied optimal estimation*", Second edition, MIT press, February 1974

[28] David Halliday, Robert Resnick and Jearl Walker, "*Fundamentals of Physics"*, 7th Edition, John Wiley & Sons, New York, 2005.

[29] Y. Kashiwagi, "*Prediction of ballistic missile trajectories*", Memorandum 37, Defense Technical Information Center, June 1968

[30] P. Zarchan, "*Tactical and Strategic Missile Guidance"*, American Institute of Aeronautics and Astronautics, Reston, Virginia, 2002

[31] Robert Hutchins, Notes for EC3310, (Optimal Estimation: Sensor and Data Association), Naval Postgraduate School, 2005, (unpublished)

[32] K.V.Ramachandra, "*Kalman filtering techniques for radar tracking*", Marcel Dekker, Inc, 2000

[33] S.S. Blackman, "*Multiple-target tracking with radar application*", Artech House, 1986

[34] J.Ferrante, "A Kalman filter-based radar track data fusion algorithm applied to a selected ICBM case", IEEE, 2004

[35] X. Yu, J. Yihui, Z. Yan, "Several methods of radar data fusion", IEEE, 2002

[36] D.L. Hall, "*Mathematical techniques in multi sensor data fusion*", Artech House, 1992

[37] A.T. Stair, J.D.Mill, "The midcourse space experiment (MSX)", proceedings on Aerospace Conference**,** IEEE, 1997

[38] D. Wright, L.Gronlung, "Decoys and discrimination in intercept test IFT-8", Union of Concerned scientists Working Paper, March 2002

[39] M.E Clark, "High range resolution techniques for ballistic missile targets", IEE Colloquium on High Resolution Radar and Sonar, May 1999

[40] X.Yu, M.R.Azimi-Sadjadi, "A neural network-based sequential bayes classifier for moving target discrimination", International Joint Conference on Neural Networks, IEEE, 1999

[41] S. E. El-khamy, F.A. Salem, "Improved Radar target identification and discrimination by matched frequency hopping spread spectrum (MFH/SS) signals and clipped radar-signature", IEEE 4th International Symposium on Spread Spectrum Techniques and Applications Proceedings, September 1996

[42] C.L. McCillough, B.V.Dasarathy, P.C.Lindberg, "Multi-level sensor fusion for improved target discrimination", Proceedings of 35th Conference on Decision and Control, December 1996

[43] "Radar tracking, Kalman filtering, & Multi-sensor data fusion" notes of Applied Technology Institute short course, 2005, (unpublished)

[44] C. Yang, S. Li, S. Mao, "Some problems of the application for multiple target tracking algorithm", IEEE, 1995

[45] S.K.Singh, M.Premalatha and G.Nair, "Ellipsoidal gating for an Airborne track while scan radar", IEEE International Radar Conference, 1995

[46] Y.Bar-Shalom, "*Multitarget-multisensor tracking: Advanced application*", Artech House, 1990

[47] D.B. Reid, "An Algorithm for tracking multiple targets", IEEE Transaction on Automatic Control, Vol. AC-24, No. 6, December 1979

[48] T. Bhattacharya, A. Preenji, T.J. Nohara, P.Weber, "Evaluation of fast MHT algorithm", RADARCON 98. Proceedings of the 1998 IEEE

[49] M. de Feo, A. Graziano, R.Miglioli, A.Farina, "IMMJPDA versus MHT and Kalman filter with NN correlation: performance comparison", IEE Proc.-Radar.Sonar Navig., Vol.144, No. 2, April 1997

[50] S. Deb, K. R. Pattipati and Y. Bar-Shalom, "A Multisensor-Multitarget Data Association Algorithm for Heterogeneous Sensors", IEEE Trans. Aerosp. Electronic Systems, AES-29(2):560-568, April 1993.

[51] K.Chang, C. Chong, Y. Bar-shalom, "Joint Probabilistic data association in Distributed sensor Networks", IEEE Transactions on Automatic Control, Vol. Ac-31, No. 10. October 1986

[52] "Exoatmospheric kill vehicle", http://www.missilethreat.com/systems/ekv_usa.html, last accessed June 09, 2005

[53] "A view to a kill", http://www.llnl.gov/str/November02/Pertica.html, last accessed June 09, 2005

[54] "Timeline: Missile defense 1944 – 2002", http://www.pbs.org/wgbh/pages/ frontline/shows/missile/etc/cron.html, last accessed June 09, 2005

[55] "Lockheed HOE", http://www.designation-systems.net/dusrm/app4/hoe.html, last accessed June 09, 2005

[56] "Lockheed ERIS", http://www.designation-systems.net/dusrm/app4/eris.html, last accessed June 09, 2005

[57] "Ground-based midcourse defense (GMD) System Exoatmospheric Kill Vehicle (EKV) data sheet", http://www.raytheon.com/products/static/node4738.html, last accessed June 09, 2005

[58] "Alternatives for boost-phase missile defense", The Congressional Budget Office, July 2004

[59] "Near field IR experiment (NFIRE)", http://www.skyrocket.de/space/doc_sdat/nfire.htm, last accessed June 09, 2005

[60] A. V. Jelalian, "*Laser Radar system*", Artech House, 1992

[61] Q.M. Lam, J.P. Hill, D. Cooke, "Improvement of laser pointing performance using a joint observer-based adaptive controller", proceedings of the American Control Conference, June 1994

[62] P.E. Pace, "*Low Probability Intercept Radar (PLI)"*, Artech House, 2003

[63] S. Josef, "On the feasibility of 'hit-to-kill' in the interception of maneuvering targets", Proceedings of the American Control Conference, AACC, 2001

[64] R.M. Lloyd, "*Physics of direct hit and near miss warhead technology*", AIAA, 2001

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, VA

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, CA

3. Dr. Dan Boger
   Information Sciences Department
   Monterey, CA

4. Dr. Phillip E. Pace
   Department of Electrical and Computer Engineering
   Monterey, CA

5. Dr. Murali Tummala
   Department of Electric and Computer Engineering
   Monterey, CA

6. Captain Kursad Yildiz
   Turkish Air Force
   Ankara, Turkey

7. Captain Kubilay Uzun
   Turkish Air Force
   Ankara, Turkey

8. Mr. Dale S. Caffall
   Missile Defense Agency
   Washington, D.C

9. Prof. Bret Michael
   Department of Computer Science
   Monterey,CA

10. Prof. Mautak Shing
    Department of Computer Science
    Monterey,CA

11. Prof. Doron Drusinsky
    Department of Computer Science
    Monterey,CA

12. LTC Tom Cook
Department of Computer Science
Monterey,CA